

AFRL-RI-RS-TR-2009-79
Final Technical Report
April 2009



JVIEW VISUALIZATION FOR VIRTUAL AIRSPACE MODELING AND SIMULATION

CACI Technologies, Inc.

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2009-79 HAS BEEN REVIEWED AND IS APPROVED FOR
PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION
STATEMENT.

FOR THE DIRECTOR:

/s/

PETER A. JEDRYSIK
Work Unit Manager

/s/

ROBERT S. MCHALE
Deputy Chief, Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE*Form Approved*
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**1. REPORT DATE (DD-MM-YYYY)**

APR 09

2. REPORT TYPE

Final

3. DATES COVERED (From - To)

Feb 04 – Nov 08

4. TITLE AND SUBTITLE

JVIEW VISUALIZATION FOR VIRTUAL AIRSPACE MODELING AND SIMULATION

5a. CONTRACT NUMBER

FA8750-04-C-0051

5b. GRANT NUMBER**5c. PROGRAM ELEMENT NUMBER**

N/A

6. AUTHOR(S)

Aaron McVay, Daniel Krisher and Patrick Fisher

5d. PROJECT NUMBER

NASA

5e. TASK NUMBER

BA

5f. WORK UNIT NUMBER

01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)CACI Technologies, Inc.
1300 Floyd Ave
Rome NY 13440**8. PERFORMING ORGANIZATION
REPORT NUMBER****9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**AFRL/RISE
525 Brooks Rd.
Rome NY 13441-4505**10. SPONSOR/MONITOR'S ACRONYM(S)****11. SPONSORING/MONITORING
AGENCY REPORT NUMBER**
AFRL-RI-RS-TR-2009-79**12. DISTRIBUTION AVAILABILITY STATEMENT**

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 88ABW-2009-1187

13. SUPPLEMENTARY NOTES**14. ABSTRACT**

This Final Technical Report discusses the accomplishments of an effort to support NASA and FAA goals for visualization of the National Airspace System (NAS) to aid in the analysis of proposed changes in order to increase its capacity to meet future needs. CACI developed a new application using JView technology called the Airspace Concept Evaluation System (ACES) Viewer. The ACES Viewer is an information visualization tool designed to provide visual representations of the output of the ACES NAS simulation developed at NASA. The application provides mechanisms to load the various types of data used and output by the ACES simulation, process the data, and then display it. The ACES Viewer is not limited to visualization of ACES data, as any data (in one of the supported formats) can be loaded and displayed.

15. SUBJECT TERMS

National Airspace System visualization, Airspace visualization, de-cluttering visualization, overlapping data visualization

16. SECURITY CLASSIFICATION OF:**a. REPORT**
U**b. ABSTRACT**
U**c. THIS PAGE**
U**17. LIMITATION OF
ABSTRACT**

UU

**18. NUMBER
OF PAGES**

134

19a. NAME OF RESPONSIBLE PERSON

Peter A. Jedrysik

19b. TELEPHONE NUMBER (Include area code)

N/A

Table of Contents

1.0	Executive Summary	1
2.0	Objective	2
3.0	Introduction.....	2
4.0	Technical Tasks	3
4.1	General JView Support.....	3
4.1.1	Infrastructure Improvements.....	3
4.1.1.1	Migration from GL4Java to JOGL	3
4.1.1.2	Geometry Buffers.....	3
4.1.1.3	Performance Improvements	3
4.1.1.4	Concurrency	4
4.1.1.5	Fault Tolerance	4
4.1.1.6	Build Process Improvements	4
4.1.1.7	Multi-threaded Rendering Support	4
4.1.2	Visual Elements	5
4.1.2.1	Generic Progress Bar	5
4.1.2.2	Histogram Grid Element.....	5
4.1.2.3	Polygon Renderer.....	9
4.1.2.4	HUD Text Renderer.....	12
4.1.2.5	Graticule.....	13
4.1.3	Camera Navigation System.....	13
4.1.3.1	Surface Orbit Navigator.....	14
4.1.3.2	World Orbit Navigator.....	15
4.1.3.3	Scroll Navigator	15
4.1.3.4	Clipping Plane Management.....	16
4.1.4	Data Access Components	18
4.1.4.1	ImageIO	18
4.1.4.2	3D Model Loaders	18
4.1.5	User Interface Components.....	21
4.1.5.1	Image List Component.....	21
4.1.6	Miscellaneous Improvements	22
4.1.6.1	JView Installer	22
4.2	Research, explore and develop De-cluttering concepts for visualization of overlapping information.....	22
4.2.1	Aggregate Visualization Concepts.....	23
4.2.2	Translucency	25
4.3	Development of the World Object.....	27
4.3.1	World Geodetic System (WGS84) Globe Object	27

4.3.2	Design of a World Object	27
4.3.3	Mercator Projection Imagery Support	40
4.3.4	Lambert Conic Projection	40
4.3.5	Raster Product Format (RPF) and Compressed ARC Digitized Raster Graphics (CADRG) Data Support.....	41
4.3.6	TerraServer-USA Data.....	42
4.3.7	Create DTEDImageGenerator TextureSet Object	44
4.3.8	NASA World Wind Imagery Support.....	48
4.3.9	Web Map Service (WMS) Imagery Support	49
4.3.10	Image Cache Object.....	50
4.3.11	Statistics Generation	50
4.3.12	Texture Reduction.....	51
4.3.13	WorldSettings Object.....	53
4.3.14	Multi-View Support.....	53
4.3.15	Relative Altitude Look Up.....	53
4.3.16	Grounding Elements	54
4.3.16.1	Grounding Policy	54
4.3.17	World Automated Test Program (WATP)	56
4.3.18	Node Change Listeners	57
4.3.19	Demo Browser – World Demo	58
4.4	Developed the initial design framework for providing visualization support to the Virtual Airspace Modeling and Simulations Project (VAMS)	59
4.4.1	Application Infrastructure	60
4.4.1.1	Visualization Components	60
4.4.1.1.1	Data Tables	60
4.4.1.1.2	Transforms	61
4.4.1.1.3	Display Containers.....	61
4.4.1.1.4	Renderers	62
4.4.1.1.5	Expressions	62
4.4.1.1.6	Controls.....	62
4.4.1.1.7	Visualization	63
4.4.1.2	Component Model	63
4.4.1.3	Concurrency	64
4.4.1.4	Extensibility	65
4.4.1.5	Logging	65
4.4.1.6	Image/Movie Capture	65
4.4.2	Data Access.....	66
4.4.2.1	Data Input.....	66
4.4.2.2	Data Output.....	67
4.4.3	Data Processing.....	67
4.4.3.1	Relational Transforms.....	67
4.4.3.1.1	Join Transform.....	67
4.4.3.1.2	Concatenating Transform.....	67
4.4.3.2	Filtering Transforms	68
4.4.3.2.1	Filter Transform.....	68
4.4.3.2.2	Time Filter Transform.....	68

4.4.3.3	Aggregating Transforms	68
4.4.3.3.1	Aggregator Transform	68
4.4.3.3.2	Grid Accumulator Transform	68
4.4.3.3.3	Force Grid Transform	68
4.4.3.3.4	Counting Transform.....	69
4.4.3.4	Special Purpose Transforms	69
4.4.3.4.1	Grid Maxima Calculator	69
4.4.3.4.2	Polygon Centroid Transform	69
4.4.3.5	Table Events.....	69
4.4.3.6	Expressions	69
4.4.3.7	Controls.....	70
4.4.4	Visual Components	71
4.4.4.1	Model Renderer	71
4.4.4.2	Line Renderer.....	74
4.4.4.3	Slice Volume Renderer	77
4.4.4.4	Splat Volume Renderer.....	79
4.4.4.5	Histogram Bar Renderer	80
4.4.4.6	Flow Field Renderer	81
4.4.4.7	Frustum Renderer.....	83
4.4.4.8	HUD Table Renderer	84
4.4.4.9	Display Containers.....	85
4.4.4.9.1	3D Scene	85
4.4.4.9.2	Tabular Scene.....	86
4.4.5	User Interface.....	86
4.4.5.1	Visualization Composition.....	86
4.4.5.2	Property Editor.....	88
4.4.5.3	Dataset Definition	89
4.4.5.4	Scene Display.....	91
4.4.5.5	Movie Capture	92
4.4.5.6	Log Viewer	93
4.5	Audit Trail Viewer.....	94
4.5.1	Graphical User Interface Improvements	94
4.5.1.1	Digital Terrain Elevation Data (DTED) Panel.....	94
4.5.1.2	Analysis View Tab.....	95
4.5.1.3	Help Panel.....	96
4.5.1.4	Geometry Panel.....	96
4.5.2	2D Graphical Engine – 2D View	97
4.5.2.1	Arrow2D	97
4.5.2.2	Trail 2D.....	98
4.5.2.3	Cone2D	99
4.5.2.4	Models 2D.....	99
4.5.2.5	Labels 2D.....	100
4.5.2.6	Imagery 2D	101
4.5.3	Splash screen.....	102
4.5.4	Representation of Ground Entities.....	103
4.5.4.1	Highlight 2D Elements	105

4.5.4.2	3D Views	105
4.5.4.3	GATER II Simulation	106
4.5.4.4	ATV and the World	108
4.6	3D Model Development.....	108
4.6.1	COTS Software Used.....	109
4.6.1.1	3D Studio Max (3DS)	109
4.6.1.2	PloyTrans	109
4.6.1.3	Multigen Creator	109
4.6.2	JView Model Software Developed	109
4.6.2.1	ElementViewer	109
4.6.3	Model Library	110
4.6.3.1	Models Developed	110
4.6.3.2	Articulated Models Developed	111
4.6.3.3	Model Thumbnails	111
5.0	Conclusion	111
	REFERENCES:	112
	APPENDIX:.....	113
	ACRONYMS:.....	123

LIST OF FIGURES

Figure 1 - Generic Process Bar	5
Figure 2 - Utilization of discrete gridded air-space over time.....	6
Figure 3 - Mesh rendering of the grid values.....	7
Figure 4 - WGS84 projected histogram.....	7
Figure 5 - Extruded Rectangles and Wireframe	8
Figure 6 - The Histogram Grid Feature with Frustum Culling.....	8
Figure 7 - Histogram Grid: Showing Force-Effect Processor Output (left), Blue Channel of Each Pixel of an Image (right).....	9
Figure 8 - Extruded Air-traffic control center zones	10
Figure 9 - Sector zones displayed at their correct (scaled) volumes using the Polygon Renderer	10
Figure 10 - Air-Zone Utilization Histograms	11
Figure 11 - Sector Zone Capacity Histograms. Sectors whose capacity has been exceeded are colored red.	11
Figure 12 - Lit Sector Zone Load-vs.-Capacity Visualization.....	12
Figure 13 - Text drawn by the HUD Text Renderer with centered justification, top (vertical) alignment, and positioned at 50% screen width and 100% screen height	12
Figure 14 - Graticule 2D	13
Figure 15 - Surface Orbit Position Model.....	15
Figure 16 - Scroll Navigator Demo Interface	16
Figure 17 - The view frustum of a camera looking toward the <i>World</i> surface. The near and far clipping planes are managed to ensure the terrain is visible.....	17
Figure 18 - Camera Clipping Planes using Bounding Box information provided by the polygon renderer (the US political boundaries).....	18
Figure 19 - ESRI Model.....	19
Figure 20 - KML Model	19
Figure 21 - Collada Model.....	20
Figure 22 - Move Demo.....	22
Figure 23 - Electric Field Created by Point Charges	24
Figure 24 - IsoSurface of an Electric Field Created by Point Charges.....	25
Figure 25 - Translucency Used to Display Multiple Visualization Elements	26
Figure 26 - Textual Labels Feature	26
Figure 27 - WGS84 Coordinate System	27
Figure 28 - Diagram of the Vertices Linked List Data Structure for Three Fan Nodes	28
Figure 29 - Actual Vertices List Structure on the <i>World</i> Object	28
Figure 30 - Unstitched Node Geometry.....	29
Figure 31 - Stitched Node Geometry	30
Figure 32 - Triangles Used to Generate Normals	30
Figure 33 - <i>World</i> Object Using Node Area Calculation.....	32
Figure 34 - <i>World</i> Object Using Screen Error Metric Calculation	32
Figure 35 - Screen Error Metric.....	33
Figure 36 - NASA's Blue Marble Lowest Resolution Image on <i>World</i> Object.....	34
Figure 37 - <i>World</i> View with WGS84 Projection.....	35
Figure 38 - <i>World</i> View with Flat Projection	36
Figure 39 - DTED Enabled on <i>World</i> Object	37

Figure 40 - DTED Enabled on <i>World</i> Object (Wire Frame)	37
Figure 41 - Multi Pass Textures.....	39
Figure 42 - Mercator Projection Coordinate	40
Figure 43 - Lambert Conic Projection	41
Figure 44 - Trinity Site with CIB 10 meter.....	42
Figure 45 - TerraServer-USA Types and Resolutions	43
Figure 46 - TerraServer-USA URBAN Imagery	44
Figure 47 - DTEDImageGenerator Original Imagery	45
Figure 48 - NASA Generated Image of North America	46
Figure 49 - DTEDImageGenerator Image using NASA Color Map	46
Figure 50 - XML Input File Example for DTEDImageGenerator.....	47
Figure 51 - DTEDImageGenerator DTED Level 0 Image displayed on <i>World</i>	48
Figure 52 - NASA <i>WorldWind</i> Imagery of Albuquerque, New Mexico.....	49
Figure 53 - WMS Image of Current Weather over NE US.....	50
Figure 54 - Texture Specific Statistics.....	51
Figure 55 - Albuquerque, NM Terra Server Data Not Limited (772 textures and 200Mb)	52
Figure 56 - Albuquerque, NM Terra Server Data Limited (211 textures and 60.5 Mb)	52
Figure 57 - Touch-Ground policy	55
Figure 58 - In-Ground policy	55
Figure 59 - Orientation definitions	56
Figure 60 - <i>World</i> Test Results	57
Figure 61 - Demo Browser – <i>World</i>	58
Figure 62 - Several Data Tables in the ACES Viewer Visualization Graph	61
Figure 63 - Several Transforms included with the ACES Viewer.....	61
Figure 64 - Two Renderers included with the ACES Viewer	62
Figure 65 - The Time Control (right) and its corresponding representation in the Visualization Graph (left).....	63
Figure 66 - A Visualization comprised of 5 displays and a variety of transforms, renderers, data tables and controls.....	63
Figure 67 - Time Query Control	70
Figure 68 – Individual Aircraft Display.....	71
Figure 69 - Maneuver Locations drawn with the Model Renderer.....	71
Figure 70 - Center Boundary Crossing Feature (Yellow Spheres).....	72
Figure 71 - Aircraft Models Rendered by the Model Renderer.....	73
Figure 72 - Location Model Feature Autosize.....	73
Figure 73 - Aircraft Model Feature with Different Rendering Properties	74
Figure 74 - Line Renderer.....	75
Figure 75 - Conflict Detection Feature (red lines), along with aircraft models, and trajectories .	75
Figure 76 - Aircraft Trajectory Visualization	76
Figure 77 - Airport Network Feature	76
Figure 78 - View Aligned Slices, side view: Wire frame (left), Textured (right)	77
Figure 79 - The final blended image from the camera perspective	78
Figure 80 - The VAMS Aircraft Density Field Visualization	78
Figure 81 - A Point Splat	79
Figure 82 - Splat Volume Renderer.....	80
Figure 83 - Histogram Bar Renderer	80

Figure 84 - Histogram Bar Renderer Configurations	81
Figure 85 - Flow Field Renderer.....	82
Figure 86 - The Flow Field Renderer drawing wind direction and intensity over Asia	83
Figure 87 - A 3D scene displaying the bounding boxes of several visualization features. Also includes a view of the camera frustum	84
Figure 88 - Table Renderer displaying the number of aircraft in each sector zone.....	85
Figure 89 - A prototype tabular scene with a visualization feature that displays the number of aircraft currently enroute to each airport	86
Figure 90 - The Visualization Composition Interface	87
Figure 91 - The Save Visualization Dialog.....	88
Figure 92 - The Property Editor.....	88
Figure 93 - The Datasets task pane	89
Figure 94 - Dataset Wizard Step 1: select the type of data to load.....	90
Figure 95 - The Basic Database Dataset Wizard Steps.....	90
Figure 96 - The Visualization Window	91
Figure 97 - The Visualization Window after activating Movie Capture	92
Figure 98 - Movie Parameters Dialog Window	93
Figure 99 - Integrated Log Viewer Window.....	94
Figure 100 - Terrain Panel	95
Figure 101 - Analysis View (Old) / (New)	95
Figure 102 - Help Panel (Old) / Help Panel (New)	96
Figure 103 - ATV Element Viewer	97
Figure 104 - Arrow 2D	98
Figure 105 - Trails 2D.....	98
Figure 106 - ATV Cone 2D	99
Figure 107 - ATV 2D Models.....	100
Figure 108 - ATV 2D Labels	101
Figure 109 - ATV with 2D Maps.....	102
Figure 110 - ATV Splash Screen	103
Figure 111- Ground Trail.....	104
Figure 112 - Triangle Selection Example	105
Figure 113 - Highlighted Elements.....	105
Figure 114 - HUD and Cockpit Views	1066
Figure 115 - ATV Keyholes	107
Figure 116 - Identifying ATV Elements.....	107
Figure 117 - ATV <i>World</i>	108
Figure 118 - Element Viewer.....	110

LIST OF TABLES

Table 1 - Method for Projection Interface	35
Table 2 - Aircraft / Surveillance	113
Table 3 - Aircraft / Bombers	113
Table 4 - Aircraft / Fighters	114
Table 5 - Aircraft / Unmanned Aerial Vehicle (UAV)	115
Table 6 - Computer Hardware / IO Devices	116
Table 7 - Computer Hardware / CPUs	117
Table 8 - Network Hardware	117
Table 9 - Communications Equipment	118
Table 10 - Ground Vehicles	119
Table 11 - Satellites	120
Table 12 - Space Telescopes	121
Table 13 - Radars	122

1.0 Executive Summary

The work for this project was apportioned into the following major areas:

Enhancement of the JView Application Programming Interface (API) – JView is a 2D and 3D runtime configurable, platform independent visualization API. JView is written entirely in Java and the 3D components utilize the OpenGL API to gain hardware graphics acceleration. Under this contract, CACI provided support to JView users resolving their technical issues, assisting new JView users, and provided general enhancements to the JView API when user requirements were identified.

Development of the *World Object* – Outdoor terrain rendering is important for a large class of Geographic Information System (GIS) applications. Interactive visualization of terrain and general complex surfaces is a difficult problem that requires large data sets to be displayed and manipulated at high frame rates. Operations such as rotation and panning must be supported so a user can examine the data in critical area, while maintaining highly accurate images. CACI developed a Continuous Level-Of-Detail (CLOD) algorithm including Frustum Culling for rendering Digital Terrain Elevation Data (DTED). The CLOD algorithm is based on the observation that 3D objects located far off in the distance may be approximated by simpler versions without loss of visual quality, thus increasing the rendering performance. The *continuous* refers to having the algorithm constantly re-compute the detail level of the 3D object depending on the distance to the camera instead of having a pre-computed set of objects to choose from. The Frustum is the area in the applications 3D scene visible by the camera, defined by 6 planes named the near plane, the far plane, the left plane, the right plane, the top plane and the bottom plane. Frustum culling is simply the process of determining whether or not objects are visible in this area and then drawing only the objects that are. This algorithm was used to create the JView *World* object which has become one of JView's most requested features resulting in an Air Force patent application [1].

Research, Explore and Develop De-Cluttering Concepts for Visualization of Overlapping Information – Another area of focused research was de-cluttering concepts. When there is overlapping information that will be visually fused, it must be de-cluttered so that it becomes intelligible to the intended audience. Presentation of that information is critical to the user's ability to logically correlate the information being displayed and to come to logical conclusions based on the data displayed. An example of this might be an air traffic controller wishing to see a display representing the regions with the greatest density of aircraft. Displaying individual aircraft and their positions would not properly convey the desired information.

Develop the Initial Design Framework for Providing Visualization Support to the NASA Virtual Airspace Modeling and Simulations Project (VAMS) – CACI developed visualization solutions for displaying and understanding the current and future implementations of the National Air System (NAS). This included JView enhancements for large screen displays, using network visualizations, and both 2D and 3D visualizations of highly multivariate data. CACI developed the Airspace Concepts Evaluation System (ACES) Viewer, which allows an individual that is

unfamiliar with the nuances of the NAS, to be able to understand and appreciate the complex decisions that are made every day by all members impacting the VAMS project.

Support of the Audit Trail Viewer (ATV) – The Audit Trail Viewer (ATV) is a replacement for IVIEW 2000 and supports visualization of many different simulation data file formats. The original ATV was developed by RISF and under this contract CACI provided maintenance, support, and development of new features for the ATV. CACI made significant contributions to the release of the ATV version 1.1 which was released on November 8, 2004, and were the primary developer of the ATV version 1.2.

3D Model Development – CACI developed a library of 3D models in both the Open Flight (.flt) and Wavefront (.obj) model formats for use by JView users. An example of a model that was developed is the RASCAL 110 aircraft used in the Characterization of the UAV Network Environment (CUNE) AFRL Project. While there are many models available on the internet for download, the majority have license issues that make them unusable by JView users.

2.0 Objective

This project developed, implemented and integrated visualization technologies that supported NASA and the Federal Aviation Administration (FAA) goals for visualization of the National Airspace System (NAS). CACI worked closely with AFRL/RISF staff in the development, research, testing, and generic improvements to the JView API and other visualization tools. This included researching and implementing concepts that enhanced JView's ability to support NASA's visualization solutions.

3.0 Introduction

Under this contract, CACI built upon the excellent work that has been done by Jason Moore and other AFRL/RISF staff and support personnel developing the JView API. JView relies on concrete Object Oriented Design (OOD) and programming techniques to provide a robust and venue nonspecific visualization environment. There are three types of modules that are created within JView: venue specific modules called facilitators that conquer the specific task of placing objects in the scene and manipulating their behavior, plug-ins that are venue nonspecific and add general functionality to the system and modules that consist of data source loaders called oddments. JView is completely data source independent and is by definition a standard visualization solution since it does not concentrate solely on environments such as space, air, ground or water. When a new data type becomes available and visualization is necessary, JView allows the programmer to quickly implement the additions. It also allows the analyst to operate in an environment in which they are familiar instead of having to learn a new interface.

In direct support of NASA and FAA goals for visualization of the NAS, CACI developed a new application using JView technology called the Airspace Concept Evaluation System (ACES) Viewer. The ACES Viewer is an information visualization tool designed to provide visual representations of the output of the ACES developed at NASA (a simulation of the national air system). The application provides mechanisms to load the various types of data used and output

by the ACES simulation, process the data, and then display it. The ACES Viewer is not limited to visualization of ACES data as any data (in one of the supported formats) can be loaded and displayed.

4.0 Technical Tasks

4.1 General JView Support

4.1.1 Infrastructure Improvements

During the course of development on the JView library, we made a number of improvements to the basic infrastructure that affects all applications that use the library.

4.1.1.1 Migration from GL4Java to JOGL

JView relies on a binding library to send OpenGL commands from JView based applications to the host system's graphics driver. GL4Java originally provided this functionality for JView. Some time ago, the maintainer of GL4Java ceased active development of the library and for several years, JView continued to rely on GL4Java. We have extended its capabilities where possible to support new system architectures but unfortunately it was not feasible to continue to maintain GL4Java, particularly to expose new graphics functionality that has been added to the OpenGL specification since version 1.4 (the last version that GL4Java supported). Furthermore, GL4Java suffered from a number of technical problems, including difficulty in correctly installing the library (a procedure which JView users had to follow prior to running any JView based applications). Developers at Sun Microsystems have implemented a new OpenGL binding library, based on GL4Java and its predecessors, called Java OpenGL Library (JOGL). Unlike GL4Java, this binding library was designed with support for future revisions of the OpenGL specification in mind, and as a Sun-backed specification, will most likely be actively developed for a significant period of time. We replaced GL4Java with the Sun Microsystems supported JOGL library and this migration allowed CACI to enhance much of the basic JView functionality using new capabilities available in JOGL and simplified the JView installation process for users.

4.1.1.2 Geometry Buffers

JOGL (The OpenGL binding library used by JView versions 1.5 and greater) makes extensive use of direct buffers to transfer data between main memory and the graphics hardware. Direct buffers are storage locations in memory that are not subject to the usual memory management policies in Java (they represent contiguous, non-movable memory regions), and can be easily accessed outside of Java (e.g. by the OpenGL driver). We have added a number of utility methods, classes, and other constructs to work with direct buffers using existing JView data types that handle common usage scenarios.

4.1.1.3 Performance Improvements

We have incorporated a number of performance improvements throughout JView, particularly in the main render loop.

4.1.1.4 Concurrency

We dramatically reduced the amount of synchronization necessary throughout JView, which in turn increases opportunities for concurrent execution of code on multi-processor machines. In many cases, the reduction in synchronization directly resulted in significant performance improvements. In addition to the changes we made to existing JView components, we added several new concurrency control utilities to JView to simplify common programming patterns when using JView.

4.1.1.5 Fault Tolerance

We added error handling capabilities to ensure that a faulty rendering component will not cause JView-based applications to crash.

4.1.1.6 Build Process Improvements

We made a number of improvements to JView's Ant based build system, including re-designing the distribution structure of JView to simplify use of the JView libraries by other developers.

All releases of JView through version 1.3 were distributed as a number of libraries (16 in JView 1.3) that allowed users to select the JView functionality developers needed for their applications, and exclude any functionality provided in JView libraries that were unnecessary. This distribution structure was useful when JView was smaller and had fewer libraries, but rapidly became difficult to manage since developers using JView would have to be aware of which libraries contained the JView functionality their application needed, and would have to explicitly add each library to their applications runtime classpath. With the addition of several third party libraries to JView supporting KML, Collada, and JOGL, use of multiple independent JView component libraries became even more difficult. To address this issue, we modified the build process to incorporate all JView classes and third party libraries into a single library file.

We have also improved the build process by moving from optional compilation, where failure of compilation of a particular component does not cause the entire build to fail, to conditional compilation, where the build environment is examined to determine what should be compiled. In addition to eliminating potentially confusing messages stating that parts of the build had failed, this has allowed information to be output that describes details of the build environment and settings.

4.1.1.7 Multi-threaded Rendering Support

Originally, JView used a single threaded rendering model, where all rendering occurred in the Abstract Windowing Toolkit (AWT) Event Dispatch Thread (EDT). This thread is also responsible for processing user input (e.g. mouse and keyboard events), and drawing Java GUI components. One user developing a JView based application was experiencing relatively low frame rates with JView scenes because the application was saturating the EDT with input and

repaint events. To address these types of issues, we have implemented support for alternate threading models for rendering in JView. JOGL by default constrains all rendering operations to the EDT; however users may pass command line options to Java to change this behavior. JOGL supports three rendering modes: AWT/EDT (the default), worker thread, and application specific mode. Worker thread mode is similar to the typical EDT threading model (it is single threaded), however it moves rendering operations from the EDT to a dedicated rendering thread. The application specific mode disables JOGL control of rendering threads, requiring the application to manage concurrency. We have added support for worker and application specific modes in JView, in addition to the EDT model that was already supported. Worker mode operates similarly to the EDT model. Application specific mode causes JView to create one dedicated render thread per Graph3D to handle rendering for the associated scene.

4.1.2 Visual Elements

4.1.2.1 Generic Progress Bar

Figure 1 illustrates a generic process bar that can be added to any JView scene. This renderer was designed to display simulation playback progress directly on the visualization display, but can be used for other purposes as well.

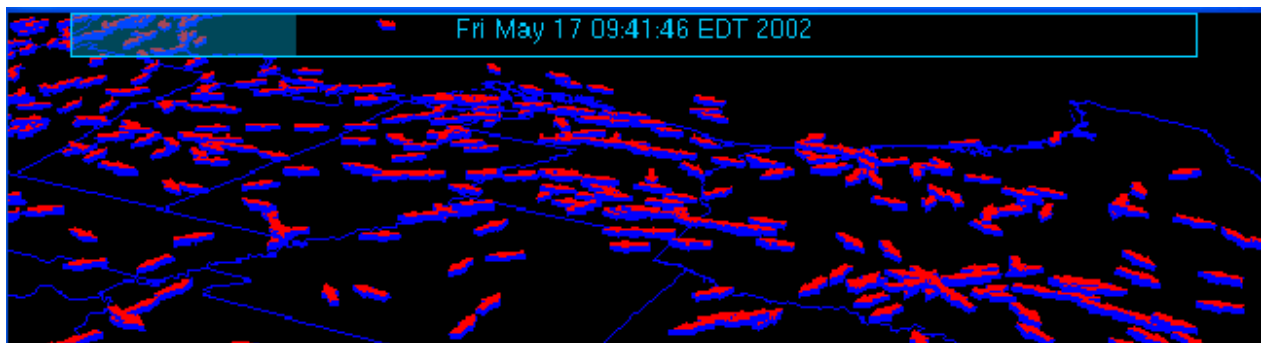


Figure 1 - Generic Process Bar

4.1.2.2 Histogram Grid Element

CACI originally developed the Histogram Grid Element to support an aggregate visualization of aircraft locations over a geographic region for the ACES Viewer. This element draws 3D histogram bars with height and coloring determined by a 2D grid of scalar values and a transfer function that maps these values to a color.

The visualization in Figure 2 uses a colored 2D rectangle for each grid cell. As the number of aircraft in a particular cell changes, the color of the rectangle changes with each color representing a density range.

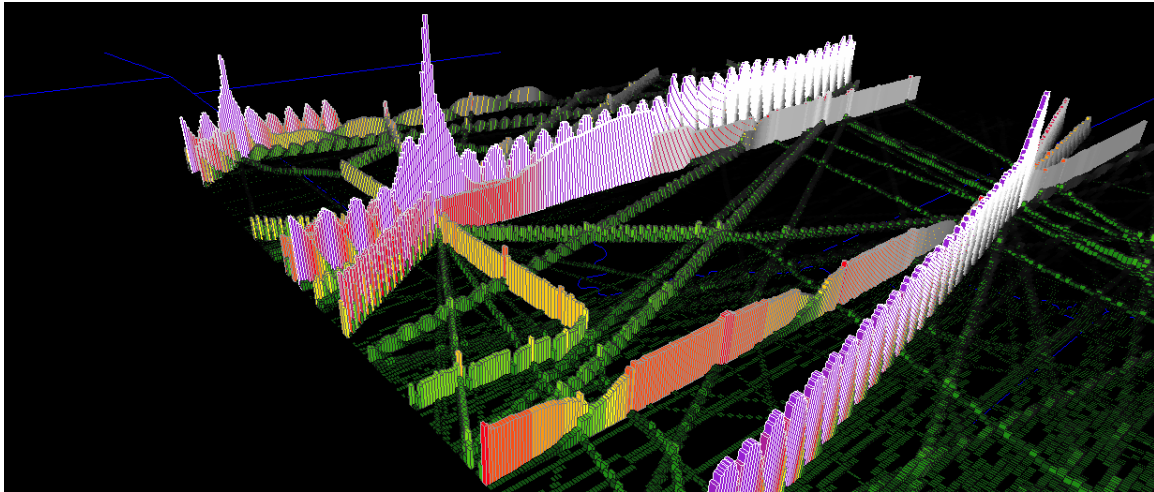


Figure 2 - Utilization of discrete gridded air-space over time.

The Histogram Grid Element provides the following capabilities:

- User specified color function. Allows for custom mapping of grid values to histogram bar colors, and has optional support for color interpolation.
- Support for histogram bar geometry or surface mesh geometry. Figure 3 shows an example of the surface mesh.
- It can be rendered using any JView Projection (Figure 4).
- Optional wireframe outline of histogram bars with a separate color function (Figure 5).
- Frustum culling (Figure 6) for improved rendering performance.
- Data independence. The Histogram Renderer requires only a 2D grid of intensity values, and specification of a region in which to draw. This allows the element to be applied to a multitude of different data sources (Figure 7 - Histogram Grid: Showing Force-Effect Processor Output (left), Blue Channel of Each Pixel of an Image (right)).

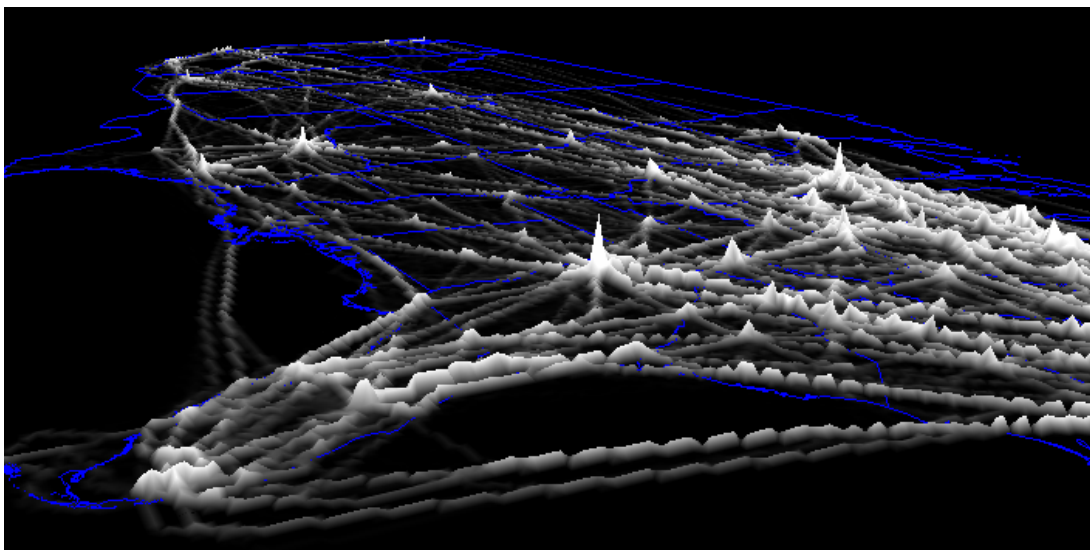


Figure 3 - Mesh rendering of the grid values.

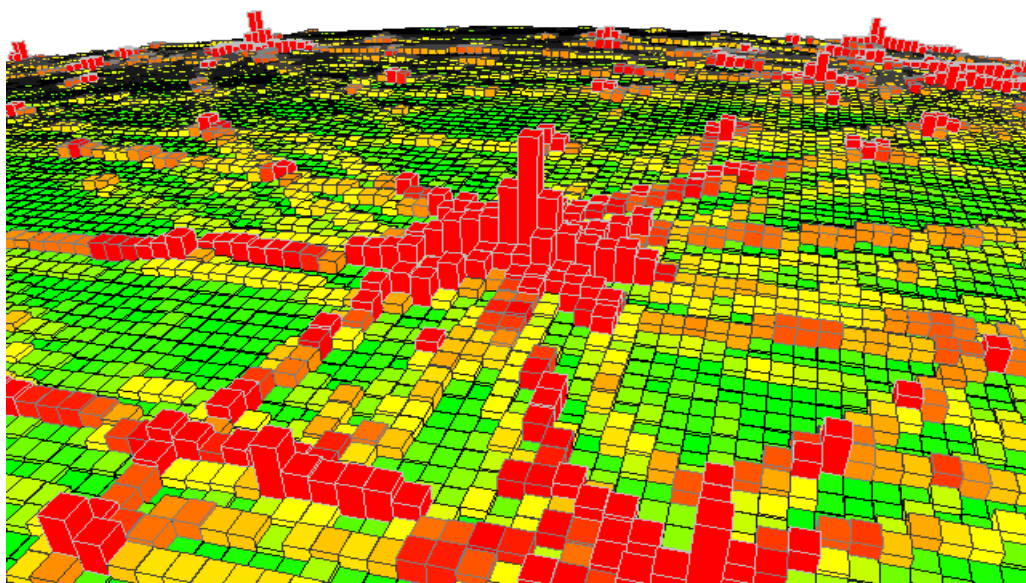


Figure 4 - WGS84 projected histogram

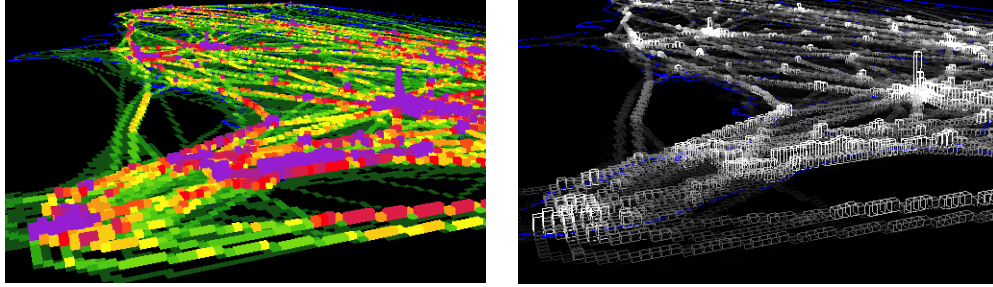


Figure 5 - Extruded Rectangles and Wireframe

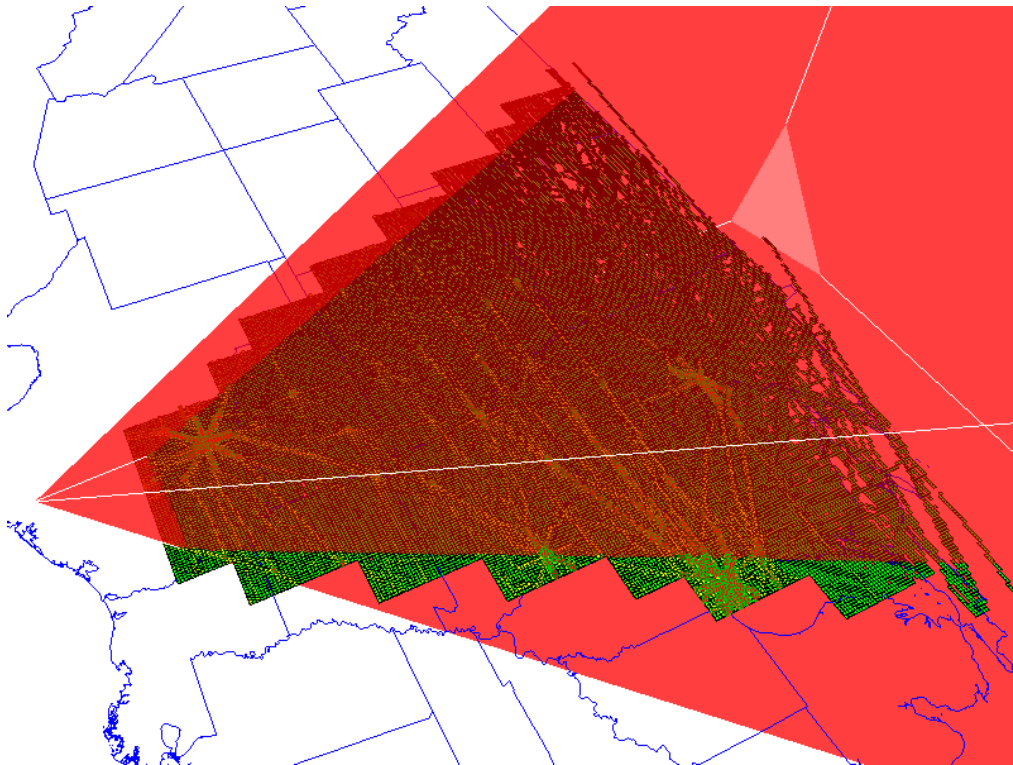


Figure 6 - The Histogram Grid Feature with Frustum Culling

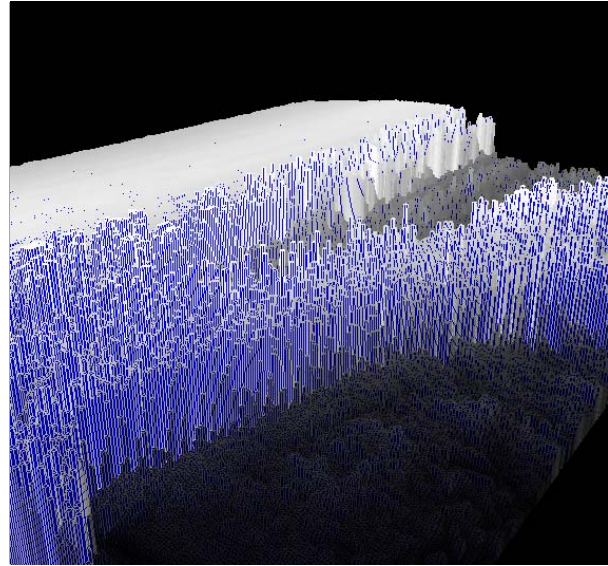
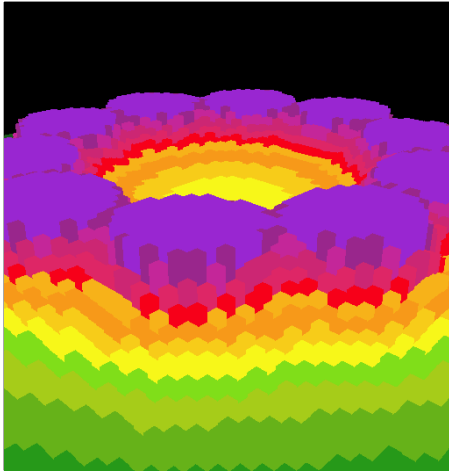


Figure 7 - Histogram Grid: Showing Force-Effect Processor Output (left), Blue Channel of Each Pixel of an Image (right)

4.1.2.3 Polygon Renderer

CACI developed a visualization component to draw 2D polygons in a 3D scene. This can be used to draw political boundaries, air zones, and other polygonal data. The Polygon Renderer has the following features:

- Polygons can be extruded to produce a 3D polygonal histogram (Figure 8), or to represent polygons that have a defined altitude range (such as the air zones of the National Air System –Figure 9).
- Per-polygon colors or materials, including support for translucency (Figure 10 and Figure 11).
- Lighting. Polygons can be lit using user specified material properties (Figure 12).

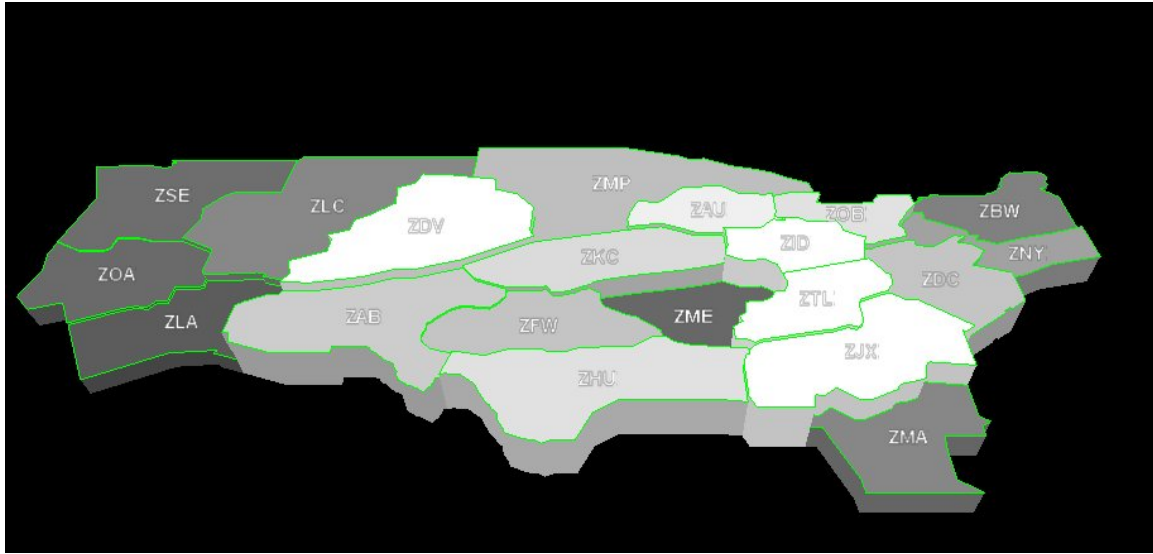


Figure 8 - Extruded Air-traffic control center zones

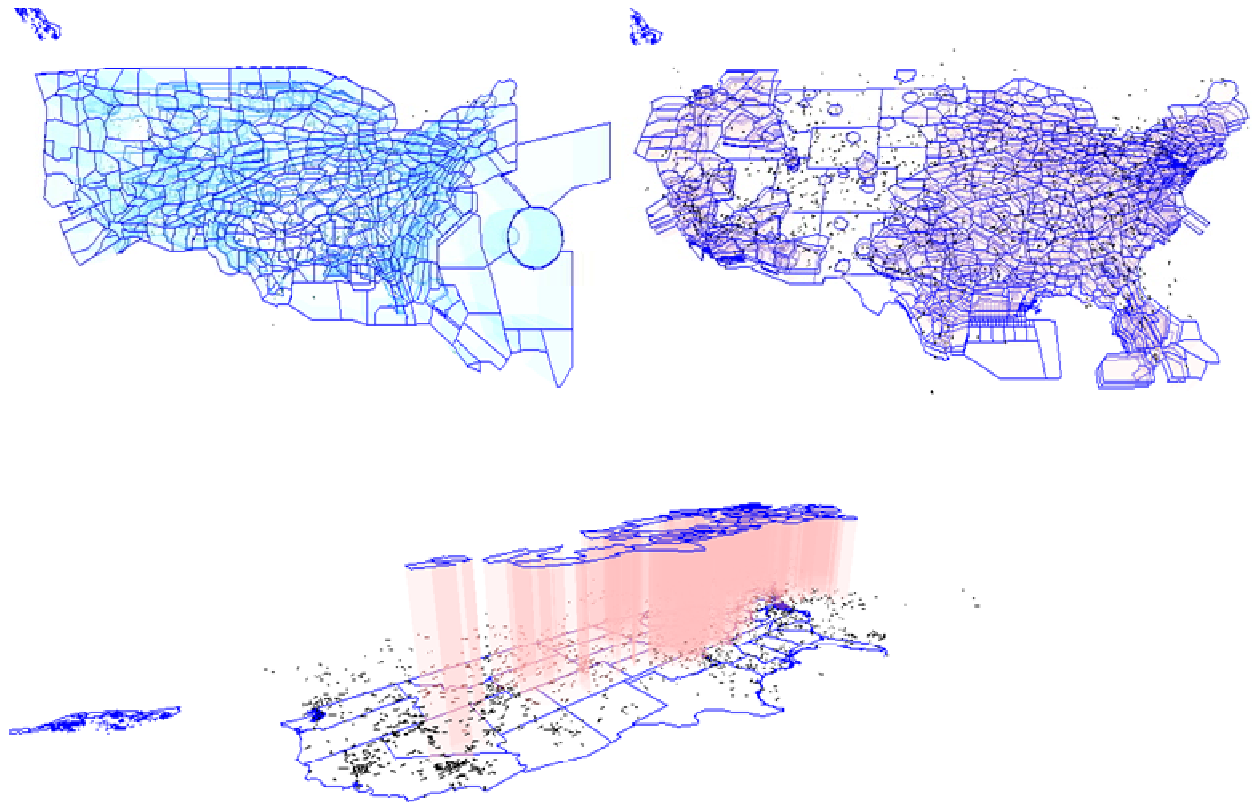


Figure 9 - Sector zones displayed at their correct (scaled) volumes using the Polygon Renderer

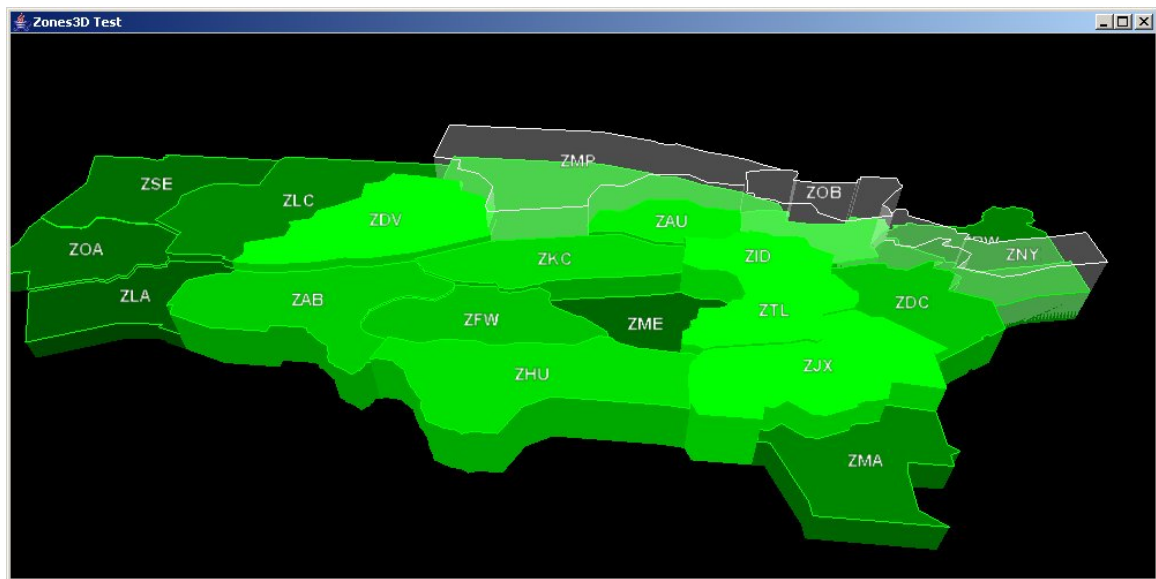


Figure 10 - Air-Zone Utilization Histograms

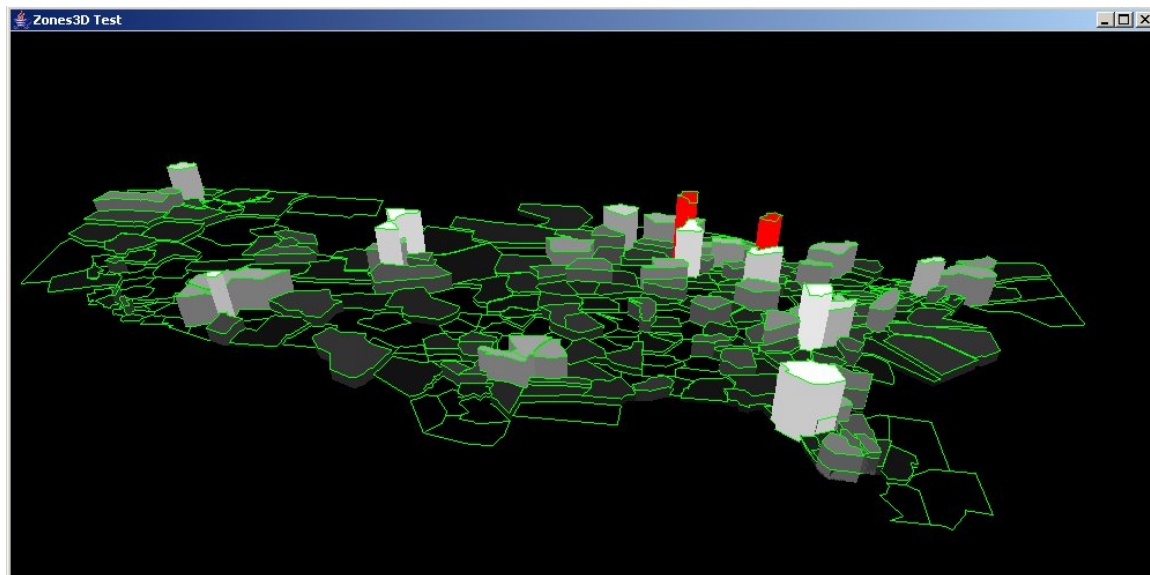


Figure 11 - Sector Zone Capacity Histograms. Sectors whose capacity has been exceeded are colored red.

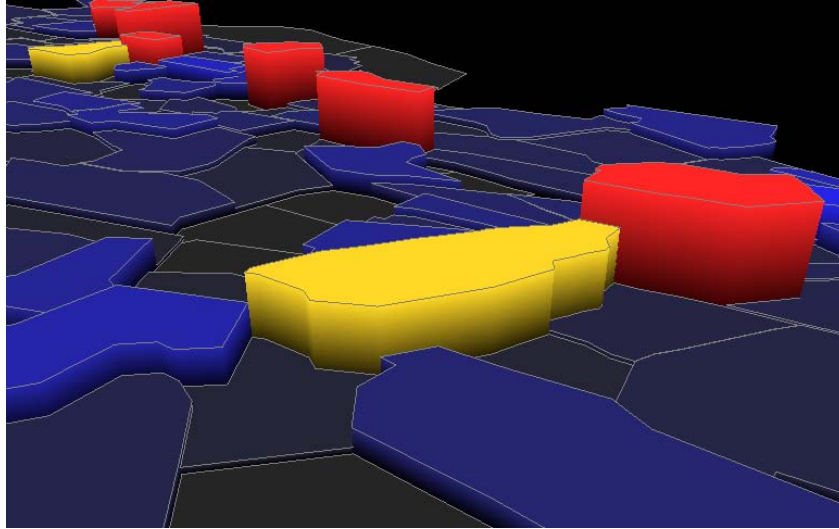


Figure 12 - Lit Sector Zone Load-vs.-Capacity Visualization

4.1.2.4 HUD Text Renderer

The HUD Text Renderer draws user specified text directly on the image plane (Figure 13). It supports:

- Configurable placement in both absolute (screen) coordinates, or as percentages of the screen size
- User defined font and coloring

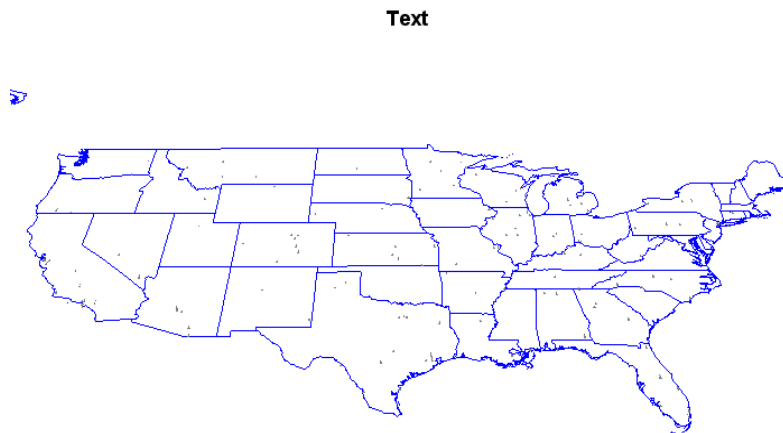


Figure 13 - Text drawn by the HUD Text Renderer with centered justification, top (vertical) alignment, and positioned at 50% screen width and 100% screen height

We have also extended this renderer to produce several informational displays, including the frame-rate, and statistics for the *World* object.

4.1.2.5 Graticule

CACI developed a Graticule package for the JView 2D engine that displays longitude and latitude lines with the addition of only 1 line of code to a JView application. The longitude and latitude lines drawn by the graticule are defaulted to values of 1 sec, 5 sec, 15 sec, 30 sec, 1 min, 5 min, 15 min, 30 min, 1 deg, 5 deg, 15 deg, 45 deg, and 90 deg. These values can be changed by the user, but were determined to be practical through a great deal of testing. The graticule determines which lines to currently draw based on the current width of the map. As the users zoom in and out, the graticule fades in and out to the next level. The Joint Synthetic Battlespace Research and Development project and the WISE project have both included this graticule in their GUI. Figure 14 illustrates the graticule in conjunction with DTED of the United States. In this example, the North 35 and 40 degree lines of latitude are currently fading in or out depending on which direction the user moves the camera from this level of detail. Latitude values above 90 degrees North or below 90 degrees South are not drawn. Longitude wraps as you move East or West past 180 degrees.

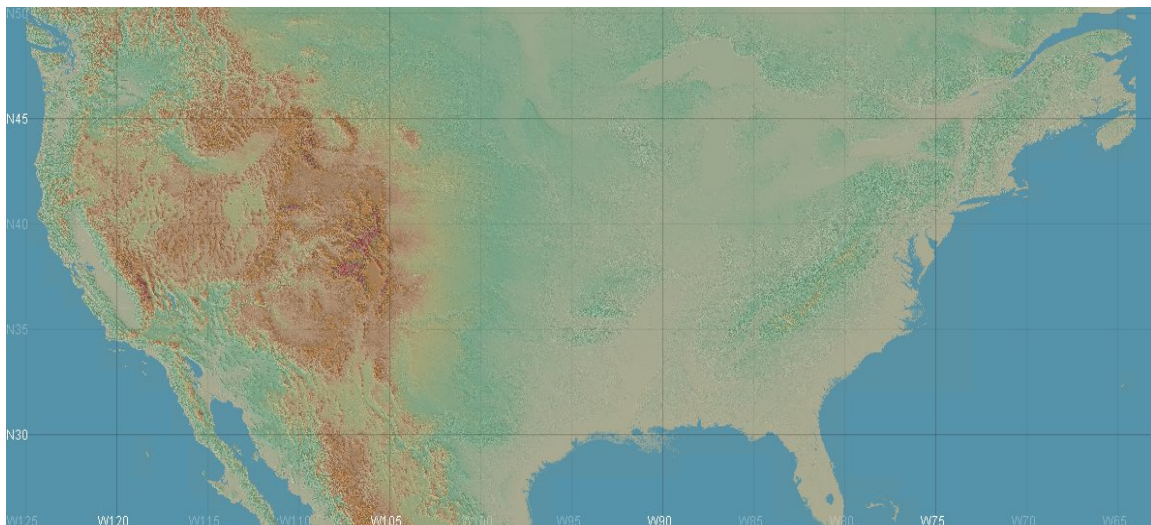


Figure 14 - Graticule 2D

4.1.3 Camera Navigation System

CACI designed and developed a new navigation system for JView that allows users to easily manipulate the camera position in a 3D scene. The navigation system is actually a framework that allows developers to implement custom navigation tools for their applications. The framework has the following features:

- A representation of position and orientation. The framework allows navigation tool implementations to define their own position model. The only restriction being that the representation can be converted to a 4D homogeneous matrix form (this is how OpenGL represents the camera position).
- A system for specifying constraints on legal positions. For example, one such constraint prevents the camera from being positioned under the terrain when a *World* element is present in a scene.
- Animations to manipulate the camera position over time.
- A clipping plane management system to automatically adjust the near and far clip plane distances based on the content of the scene.
- User input handling. This maps user inputs to the actions that the navigation system should take in response. For example, dragging the mouse in a scene might activate an animation that moves the camera position accordingly.

We have implemented several tools built upon this framework that provide different methods for controlling the camera position.

4.1.3.1 Surface Orbit Navigator

We developed the *Surface Orbit* position model, illustrated in

Figure 15, to support navigation in scenes that present geographic information. The *anchor* in this figure is a look-at point; the camera will always be oriented to face this location. The anchor location can be moved along a surface defined by a projection (a WGS84 projection in the figure) by clicking and dragging the surface with the mouse. The camera can orbit the anchor with another mouse gesture, providing different viewing angles and distances relative to the anchor.

Camera motion is handled with animations that include smooth acceleration/deceleration, arc transitions (the camera moves away from the surface before moving closer again when moving across large distances), and several other features that improve the appearance and continuity of the scene when moving around. Most of the math used to calculate position and orientation after moving the camera is based on quaternions, which avoids gimbal-lock type problems that other navigation systems (Google Earth, NASA *WorldWind*) have near the poles (latitude = +/-90 degrees).

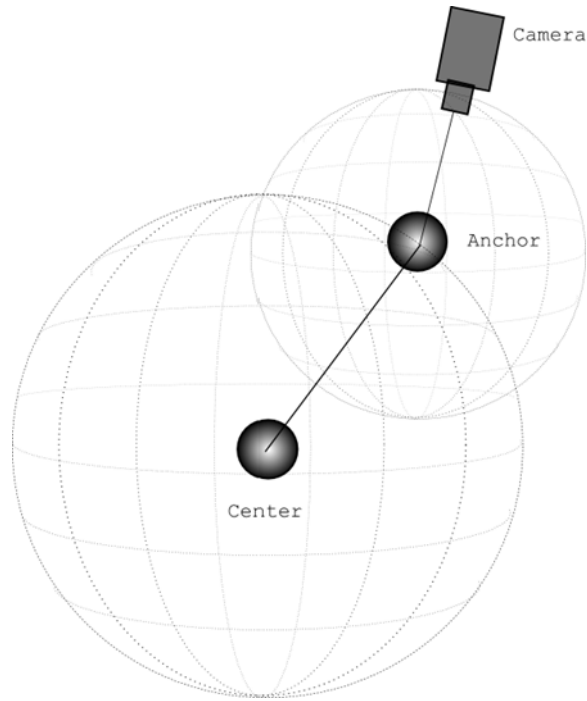


Figure 15 - Surface Orbit Position Model

4.1.3.2 World Orbit Navigator

The *World* Orbit Navigator is an extension of the Surface Orbit Navigator that can be used with scenes containing a *World* terrain renderer. This provides additional functionality over the Surface Orbit Navigator, such as taking terrain elevations into account when moving the anchor. The *World* Orbit Navigator also includes a default set of position constraints in order to prevent the camera from moving under the terrain for example as well as several other *World*-specific features.

4.1.3.3 Scroll Navigator

A third navigation tool built on top of the Camera Navigation Framework uses a traditional 2D navigation interface to manipulate the camera position in 3 dimensions. The Scroll Navigation Model represents the camera position in relation to an axis-aligned bounding box, and provides five degrees of freedom:

- Tilt and Pan to control the camera's orientation.
- Three axes of movement, specified as percentages of the corresponding view-aligned axes of the bounding box (horizontal, vertical and depth).

This model is designed to be controlled with a set of scroll bars allowing users to scroll the scene contents up/down, left/right, and in/out, in addition to turning the camera. A prototype user interface is shown in Figure 16.

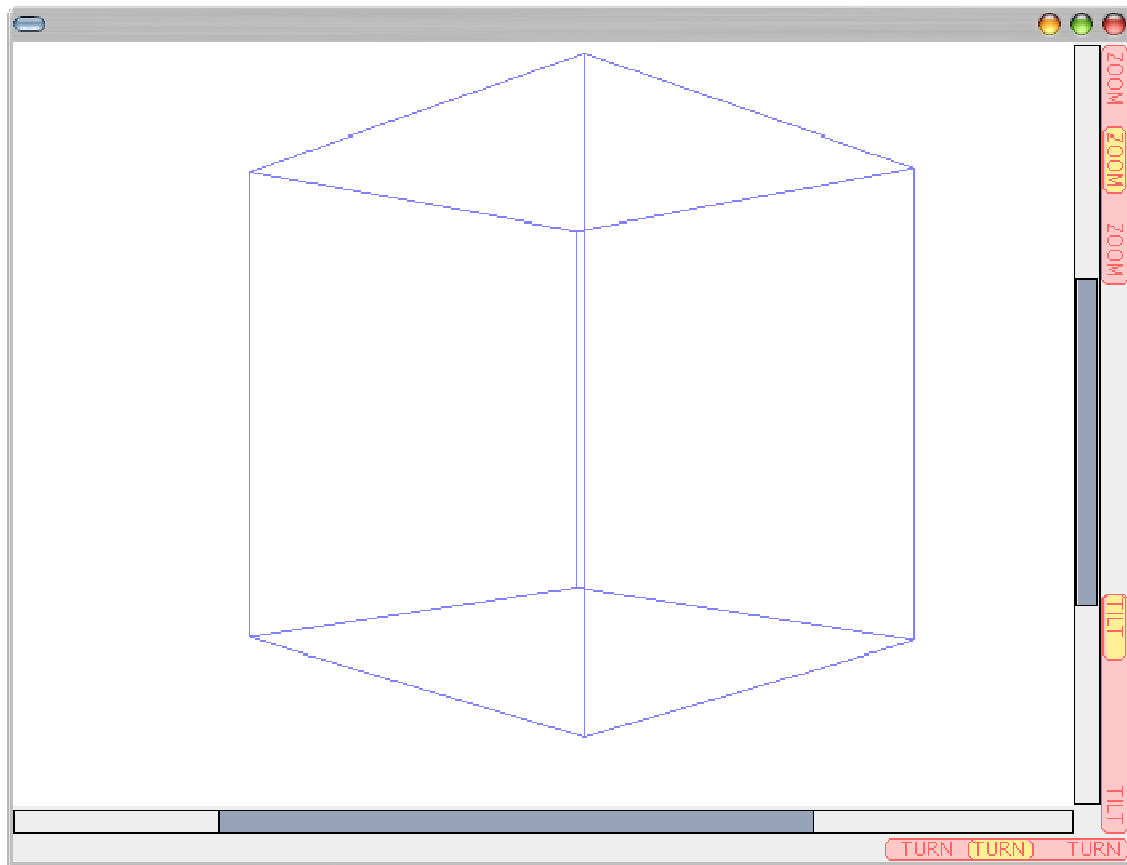


Figure 16 - Scroll Navigator Demo Interface

4.1.3.4 Clipping Plane Management

The Camera Navigation Framework includes support for automatic clipping plane management. Clipping planes define which objects in a scene are visible from a given camera viewpoint. Most scenes use 6 clipping planes. Left/right and top/bottom planes are defined automatically based on the viewport size and the field of view. The near and far clipping planes define the depth range (relative to the camera location) in which elements will be visible. Objects closer to the camera than the near plane, or farther from the camera than the far plane will not be visible. Due to limited floating point precision on graphics hardware (typically 32 bits), the depth range should be kept as small as possible to avoid depth aliasing artifacts. The clipping plane management system is designed to minimize this depth range, while still keeping all objects that the camera is looking toward visible.

Developers are free to implement and incorporate their own clipping plane management algorithms, but we provide two that should be sufficient for most applications. One is designed for use with the *World* element to ensure that the terrain is visible within the clipping plane volume as shown in

Figure 17. The other uses bounding box information provided by compatible scene elements to maintain visibility of these elements as shown in Figure 18.



Figure 17 - The view frustum of a camera looking toward the *World* surface. The near and far clipping planes are managed to ensure the terrain is visible.

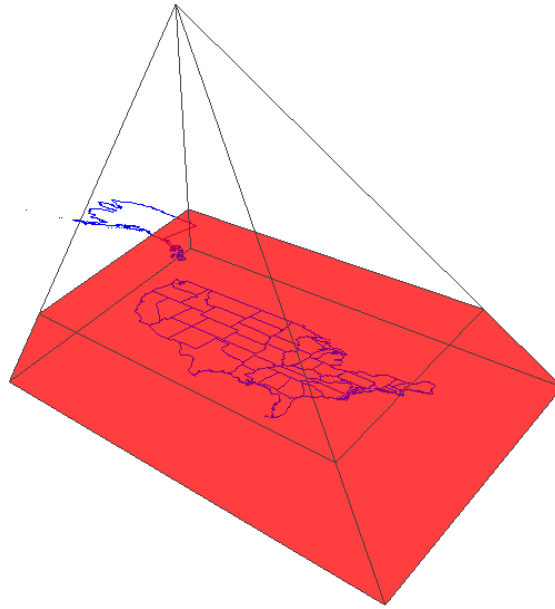


Figure 18 - Camera Clipping Planes using Bounding Box information provided by the polygon renderer (the US political boundaries).

4.1.4 Data Access Components

4.1.4.1 ImageIO

JView's ImageIO utility is used throughout JView to load texture images from disk. We have corrected several problems with URL and filename handling that was preventing images from being loaded if the filenames contained white space characters.

4.1.4.2 3D Model Loaders

CACI has written a few 3D model loaders in support of the JView contract in order to create more visual appealing models. Models contain geometry information that can be parsed and displayed by JView. The most difficult part of displaying the model is parsing the complex model structure to extract the information in a useful manner. The loaders parse information out and create the JView scene element or geometry so that users can show their model. With more model loaders available, it is easier for users to import their information or find useful models for their task. ESRI, KML and Collada Models are some of the model loaders that CACI has written. A description of each follows:

- **ESRI Model** - ESRI shape files that contain two dimensional geometric data describing a series of shapes: *Line*, *Polygon*, *etc.* ESRI files can show political boundaries, roads, and more. ESRI sparked research in Level of Detail algorithms for two dimensional lines because ESRI shapes can contain so many lines (Figure 19)

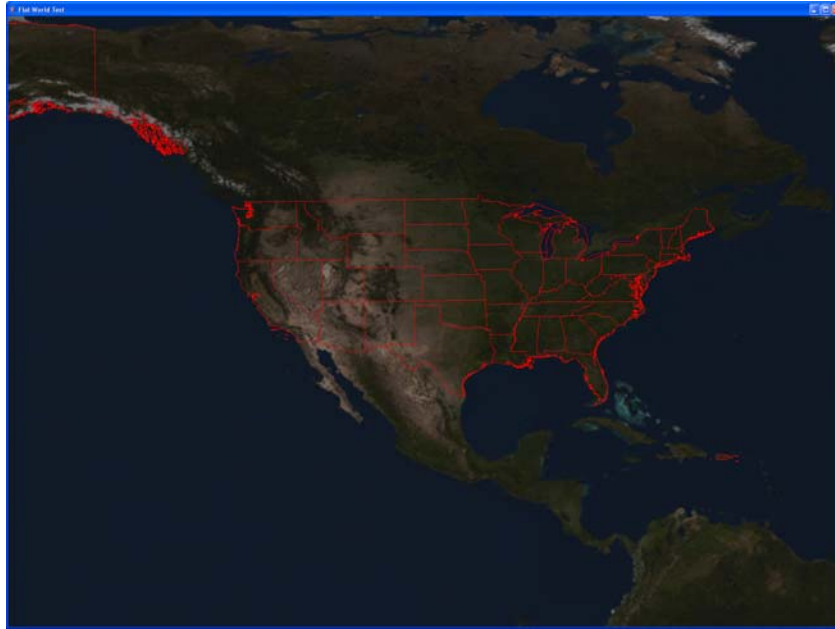


Figure 19 - ESRI Model

- KML Model - Keyhole Markup Language (KML) is an XML file that is used by Google Earth to display geometry, points, and information on a globe. Since Google Earth is so well known, KML is widely used and there are many KML files ready to display. In order to support KML, there were many other graphical objects like callouts that needed to be created. Callout are informational windows used to display information in GL (Figure 20).

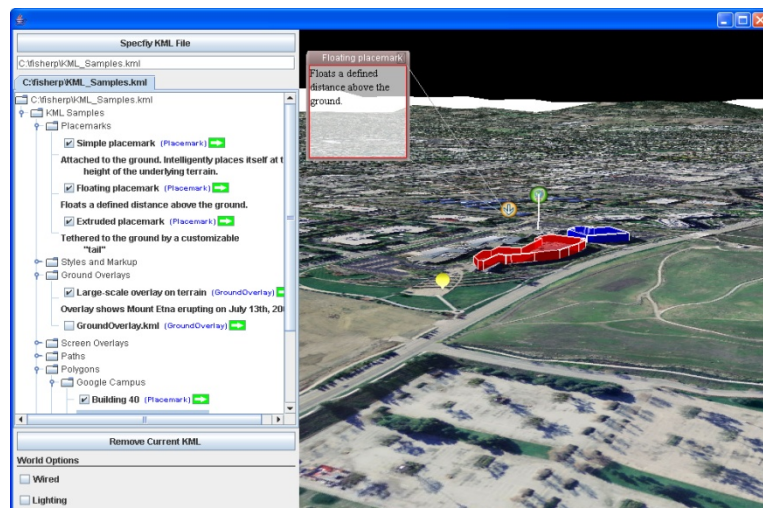


Figure 20 - KML Model

- Collada Model - The Collada model is an XML based file format that describes the geometry, textures, and shape of a 3D model. The primary reason for supporting the Collada format is that it seems to be becoming the new industry standard data asset exchange format and it is supported by a range of modeling applications. These include 3dsMax, Maya, and Google's sketch up. Since Collada is a simple XML format, it very easy to write an exporter for most applications. Supporting Collada allows JView users to obtain thousands of models (Figure 21).

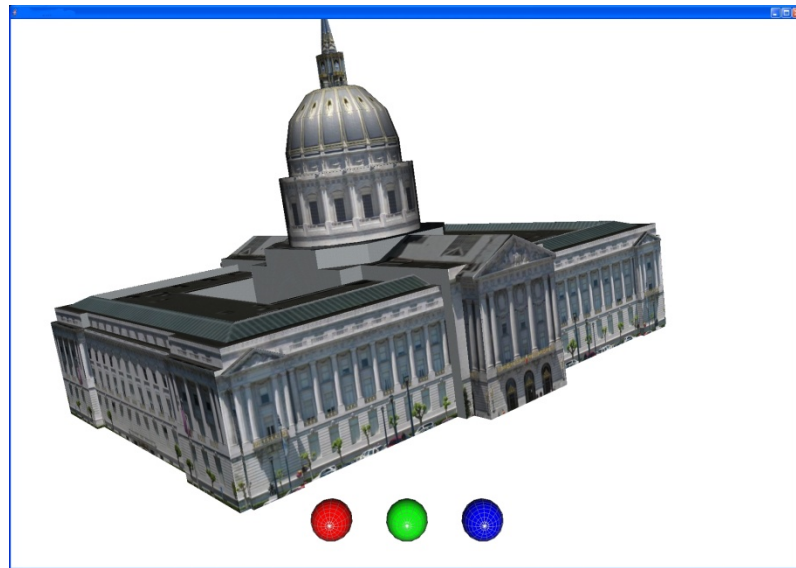


Figure 21 - Collada Model

In order to support the *parallel to ground* clamping policy for *World* elements, we designed and implemented an algorithm to calculate a composite orientation relative to a plane. Given an element's orientation (in terms of yaw, pitch and roll) interpreted as relative to an arbitrary plane, the algorithm calculates a new absolute orientation composed from the input orientation and the orientation of the plane. The original yaw of the element is preserved so that if the element had a north facing orientation, the composite orientation would still yield a north facing element when applied. The algorithm calculates a pitch and roll for the plane, and offsets the element's orientation using quaternion math. This functionality is now included in JView as it may be generally useful even outside the scope of *World* elements. This work also resulted in significant optimizations to various quaternion math utilities in JView that in turn improved the performance of camera positioning and projection calculations.

4.1.5 User Interface Components

4.1.5.1 Image List Component

The ATV has for some time included an excellent user interface component that allows users to select a 3D geometry from a list of images (typically to assign a model to a particular element in a 3D scene). We have improved this component to facilitate general use with other applications, and incorporated it into the JView controls library. The new component is made up of three parts that can be used together or separately:

- An asynchronous image loading utility. This allows images to be loaded from disk or network devices outside of Java's event dispatch thread (thus preserving interactivity of user interface components), and contains a number of tunable parameters controlling threading characteristics and image loading rate.
- A user interface component to display the images in list form. This component uses the image loading utility to load a limited number of images before redrawing, preserving interactivity.
- A list model interface to specify the contents of the list. The interface requires implementations to provide information regarding which images to fetch for each element in the list, without specifying what the images represent. This decouples the user interface component from the backing data sources. We have implemented a geometry list model to provide the functionality that the geometry list was originally developed for.

Move Demo – CACI developed a demo that randomly moves a number of elements shown in Figure 22. This demo was created to show JView's capabilities as well as test for improvements and bugs. This demo helped to show that the performance of JView's 2D engine was inadequate for drawing thousands of objects. As a result, the JView AFRL staff modified the 2D elements to allow faster draw times than before.

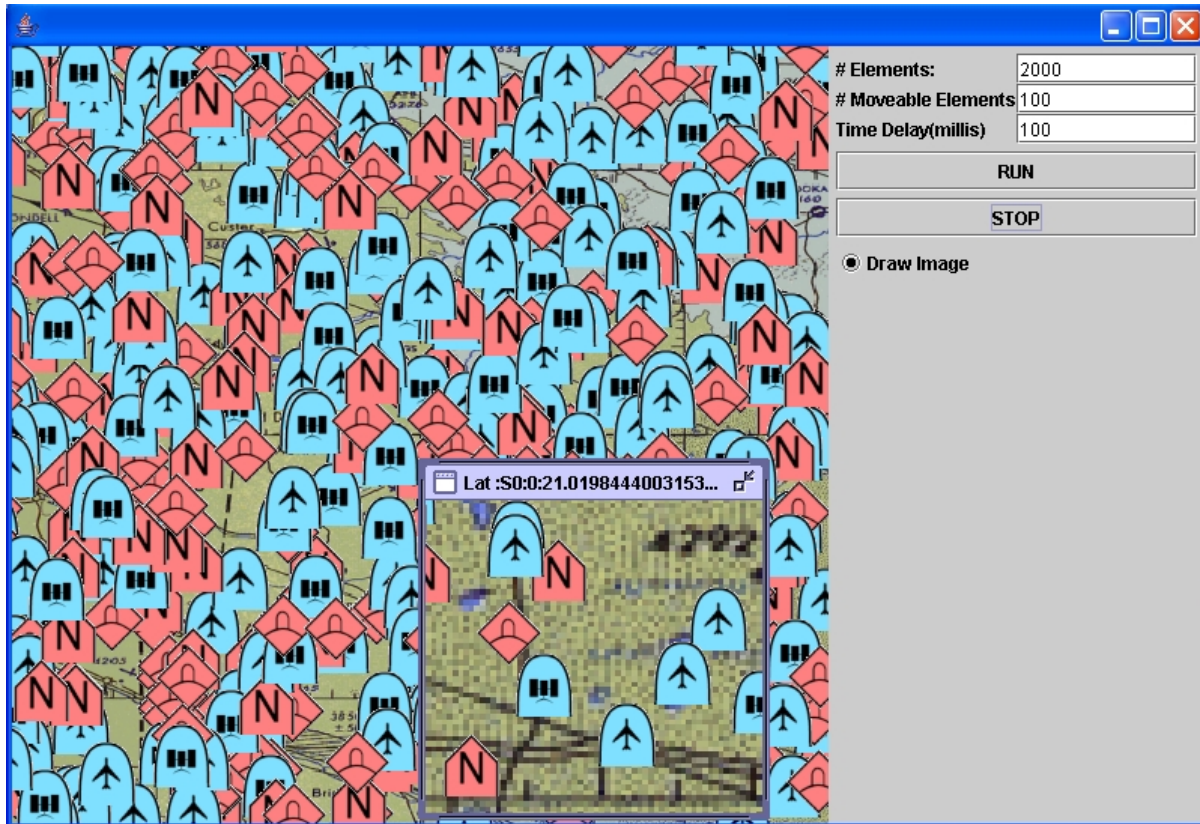


Figure 22 - Move Demo

This demo sparked another idea of testing a 2D point element. The Point2D class was created for this demo and was used to test how quickly it could perform. Currently, accurate timing of the 2D engine is not possible; this will be looked at more in the future.

4.1.6 Miscellaneous Improvements

4.1.6.1 JView Installer

CACI built the JView Installer using the InstallAnywhere application. InstallAnywhere is a Java application that makes windows installers for other Java projects and applications. The JView installer comes with three pieces that can be installed together or individually. They are DemoWorld, JView, and the DemoBrowser. InstallAnywhere experiences problems when dealing with projects that exceed two gigabytes. In order to mend this problem, we wrote a script and java program that extracted the *World* data upon installing.

4.2 Research, explore and develop De-cluttering concepts for visualization of overlapping information

Another area of focused research is de-cluttering concepts. When there is overlapping information that will be visually fused, it must be de-cluttered so that it becomes intelligible to

the intended audience. Presentation of that information is critical to the user's ability to logically correlate the information being displayed and will amount to logical conclusions based on the data displayed.

4.2.1 Aggregate Visualization Concepts

Several concepts (stated below) were explored to use aggregate information to construct a visual representation of data too voluminous to display directly.

- Developed an algorithm to divide a region of 2D space containing many different objects into a grid of user defined size. This algorithm uses the locations and heading of each object contained in the region to calculate the density, cumulative density, average heading and cumulative heading in each cell of the grid. This provides fast and simple means for condensing a large number of individual objects into a small number of groups. This algorithm is limited by the fact it creates a very block oriented picture of the region represented, and its speed is affected by the size of the grid. This algorithm was used to development several different visualizations of the National Air System (NAS) for the Virtual Airspace Modeling and Simulation Project (VAMS). These visualizations are discussed in the next section.
- Developed a polygon zone algorithm to divide a region of 2D space containing many different objects into a group of polygon zones. This algorithm used the locations of each object contained in the region to calculate the density of each polygon zone. This algorithm does not have the problem of creating a block representation of the region as in the grid algorithm, but is instead limited to the number of zones the region is broken into, and its speed is affected by the number of points defining the polygon zone. It uses the PolyUtils point in polygon test to determine how many objects are contained in each polygon zone, and it is for this purpose the PolyUtils package was originally created. As with the grid this algorithm was used to develop several different visualizations of the NAS for the VAMS Project, and these visualizations are discussed in the next section.
- Developed a metaball algorithm for calculating object density in a region of 2D or 3D space. Metaballs is a method of displaying the effect that objects have on other objects near them. The effect is simulated using the equation for the electrical force between two charged particles $F = Q / (R * R)$. An image of an electric field created by some point charges is shown in Figure 23. The strength of the field is indicated by the brightness of the point, and the points are displayed in purple.

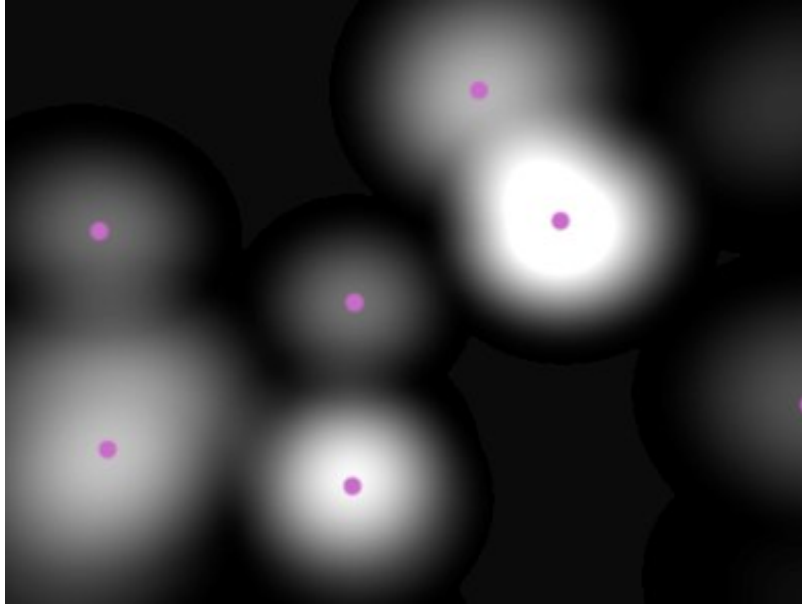


Figure 23 - Electric Field Created by Point Charges

The net electric field between all particles in the region is computed at every point along a grid dividing up the region. This grid could then be displayed by drawing an isosurface. Where you see a contour this is an equipotential line, a line along which the potential is equal at all points. Figure 24 displays an isosurface version of Figure 23.

While this method is limited by the finite resolution of the grid, the actual equation of the field has infinite resolution. Initial implementation used the same value of Q , the charge of the particle for all objects. The algorithm will be modified to include individual values of Q for each object to allow some objects to have a greater effect than others.

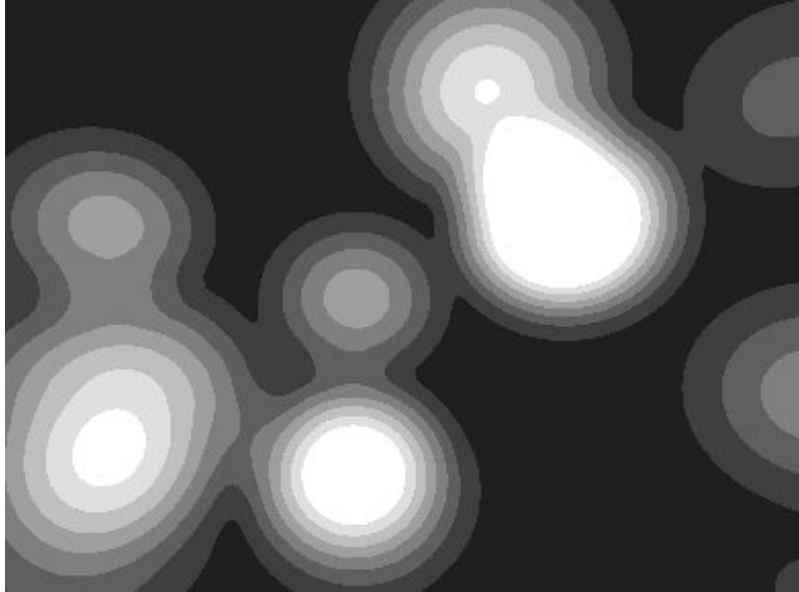


Figure 24 - IsoSurface of an Electric Field Created by Point Charges

4.2.2 Translucency

- For the visualization components of the ACES Viewer application to convey the necessary information, it is expected that several visual features will be combined in some manner both in a single display, and in multiple views of an ACES simulation run. For this to be effective, some visual de-cluttering is necessary to prevent important information from being hidden.
- The visualization shown in Figure 25 used translucency to combine three elements. They are an ARTCC usage histogram, a grid cell usage histogram and a vector map of the United States. They were constructed with no modification to existing code, only changes to feature parameters.
- Similar techniques were employed to de-clutter visualizations containing a large amount of text. The usefulness of textual labels in a visualization suffers more than most other visual elements when highly cluttered because it quickly becomes illegible as shown in Figure 26. We implemented an occlusion based de-cluttering algorithm to make the text translucent when occluded by another label.

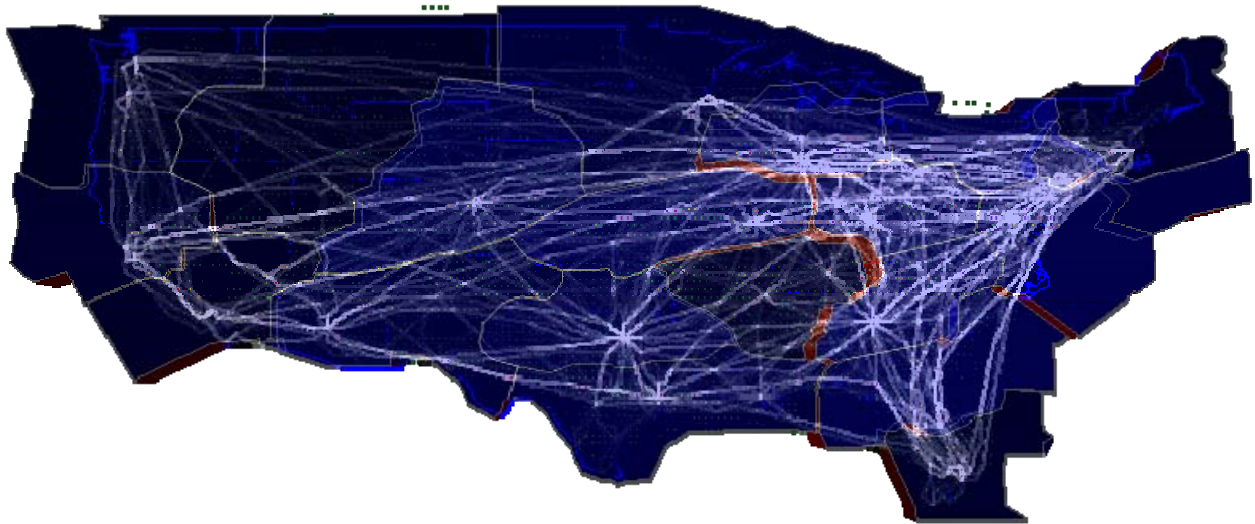


Figure 25 - Translucency Used to Display Multiple Visualization Elements

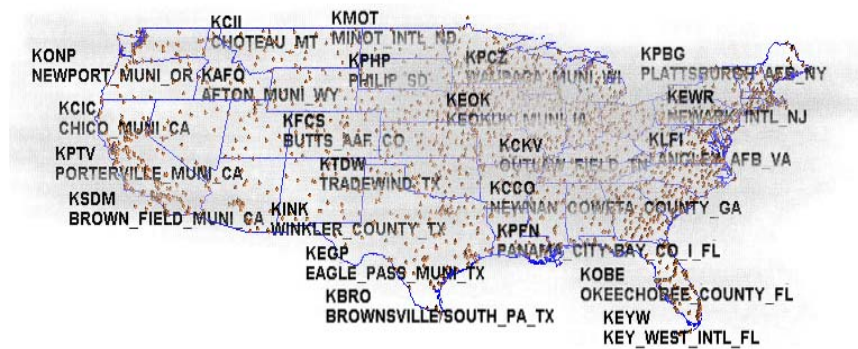


Figure 26 - Textual Labels Feature

4.3 Development of the World Object

4.3.1 World Geodetic System (WGS84) Globe Object

Figure 27 displays the 1984 *World Geodetic System* (WGS84) Coordinate System for the Earth.

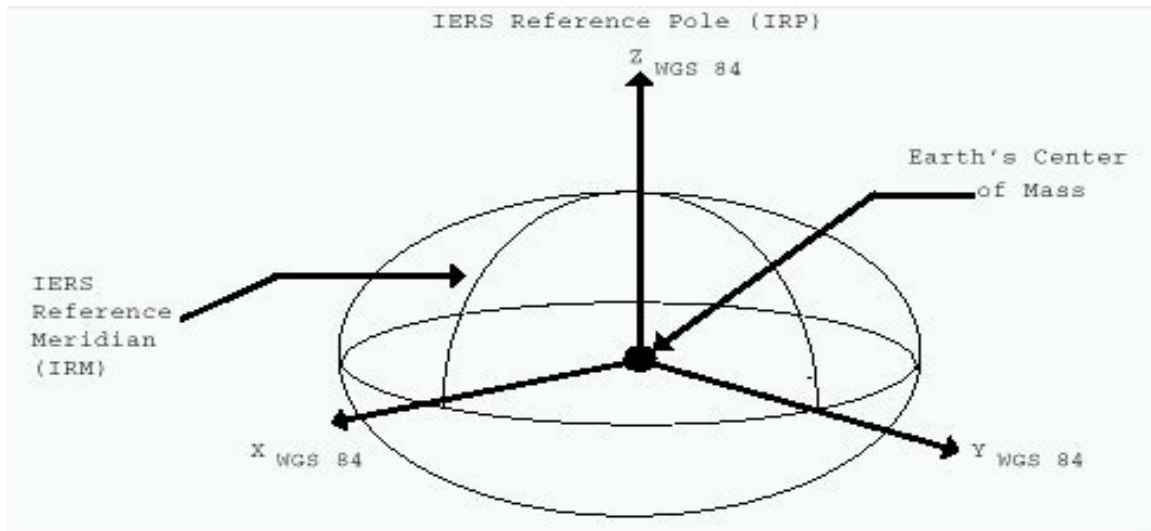


Figure 27 - WGS84 Coordinate System

4.3.2 Design of a World Object

As part of this contract, CACI developed a *World* object from the following list of requirements:

- Level of Detail for both Geometry and Imagery.
- Dynamic loading of Imagery and DTED as needed.
- Imagery not tied to Geometry, Multipass Texturing.
- Dynamic Stitching of Geometry at different levels, no shelf's.
- Frustum, Blackface, and Occlusion Culling.
- Support for multiple projections, Flat Earth, WGS84 Ellipsoid.
- Lighting.

With the desired data structure at *draw time* as the driving factor in the algorithm design, Aaron McVay (CACI) and Jason Moore (AFRL) developed a data structure and an efficient method of building the data structure for this object that is patent pending [1]. Since the vertices themselves are actually a two-dimensional grid that wraps around at the -180W and 180E longitude a two-

dimensional circular linked list of vertices was chosen to represent the draw time data structure. The list stores fan node vertices, with links to their north, south, east, and west neighbors. It was also necessary to be able to determine which of the vertices on this list formed the center points of individual fan nodes. Rather than add a flag to each entry on the list indicating it was a *center*, an additional one-dimensional list of center vertices was also created. Figure 28 displays a diagram of an example of the vertices linked list for three fan nodes. The red vertices are fan node centers and references to them appear in both the center list and the vertices list.

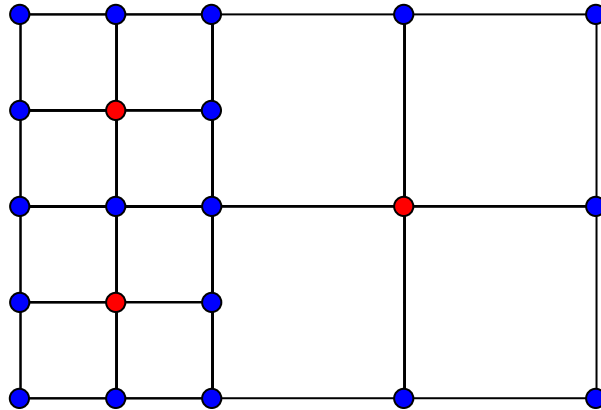


Figure 28 - Diagram of the Vertices Linked List Data Structure for Three Fan Nodes

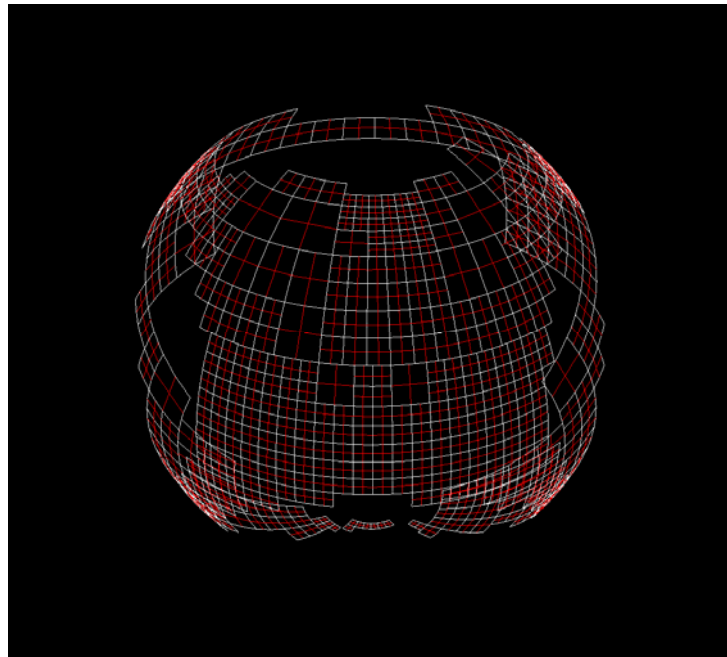


Figure 29 - Actual Vertices List Structure on the *World* Object

The vertices and center lists must be created in the step prior to the draw step, and then at draw time the center list is used to obtain entry into the vertices list (Figure 29). At this point the data structure is independent of the type of data used to create it, and would lend itself to any 3D object that requires a CLOD algorithm. The actual method of building the data structure for the *World* object follows:

Stitching the geometry together – In many CLOD algorithms such as the one used by Google Earth border nodes at different levels of detail drew shelf's to hide the fact that the geometry is not connected. Figure 30 shows an example of this problem.

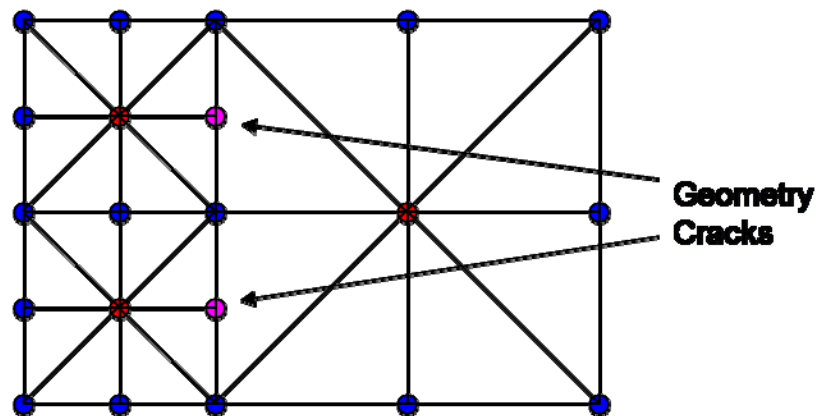


Figure 30 - Unstitched Node Geometry

In order to solve the problem in this example, the left most triangles in the larger fan node need to be broken into smaller triangles. With *JView Worlds* vertices list it was possible to add more than nine points to each fan node in order to stitch these regions together. This is accomplished by starting each node at the center point determined from the center list, then walking the vertices list to determine all vertices that border the center. Figure 31 shows what the larger fan node will look like when drawn.

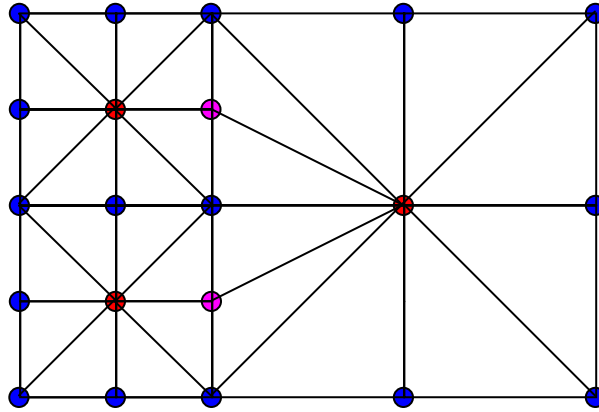


Figure 31 - Stretched Node Geometry

This approach allows for an unlimited number of fan points off each center vertices. A disadvantage is that in regions where the difference in level between nodes is very large, several long thin triangles are created. OpenGL does not like long thin triangles, but in practice this was not found to occur often enough to present a problem.

Calculating Lighting – In order to use lighting, every vertex must have a normal vector defined that is an average of all triangle normals that the vertex is a part of. Figure 32 shows the correct triangles used to calculate normals for the two pink vertices. The average of the normals for triangles 1, 2, 3, and 4 is used as the normal for the top pink vertex, and triangles A, B, C, and D are used for the bottom pink vertex.

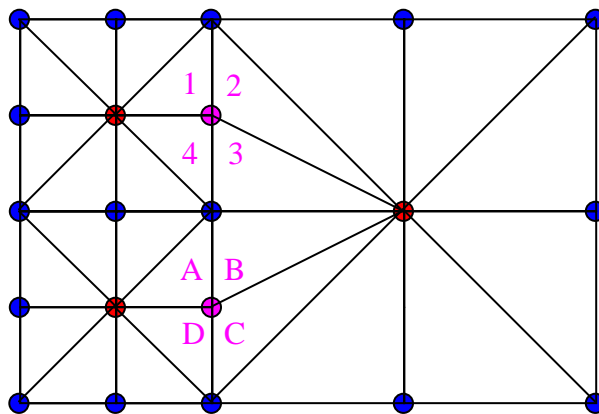


Figure 32 - Triangles Used to Generate Normals

In order to perform the normal calculation, an additional step is required preceding the draw step in which the normals are calculated for each vertex. This process starts with each center vertex, walks the fan node and sums up all normals for each vertex. During the draw step each vertex's normal summation is *normalized* resulting in the correct value for that vertex.

QuadTree Forest – The JView *World* object uses a QuadTree data structure generated dynamically at run time and only contains the top most nodes and their children that are currently within the viewing frustum. The base unit of the QuadTree is longitude and latitude post numbers. Typical globe applications break the earth up into 45 square degree sections for simplicity, but since this object uses DTED in its raw, unaltered form, the top QuadTree nodes had to reflect the layout of DTED data. DTED contains square data files containing an equal number of points from latitudes 50S to 50N. Above 50N or below 50S DTED is broken into zones that are not square, decreasing the number of longitude posts as latitude approaches the poles. No DTED was available above 82N or below 82S. To match this data the QuadTree Forest is created out of 222 different QuadTree branches symmetrically covering all but the poles. The bottom most level of detail for the QuadTree is based on 3601x3601 per 1 square degree to support up to DTED level 2. The zero points for both longitude and latitude posts are defined to be at 180W and 90S. Using a consistent base size of 3601x3601 for every square degree allows the vertex linked list to be symmetrical across the entire globe. This requires the QuadTree object to map its actual post range based on the DTED zones to this 3601x3601 per degree base size for the vertex list.

Resolution Metric – The final implementation of the *World* Object on this contract supports the *Node Area* and the *Screen Error* Metric options for determination of level of detail. Figure 33 shows the *World* Object using the Node Area Calculation, and Figure 34 shows the Screen Error Metric method.

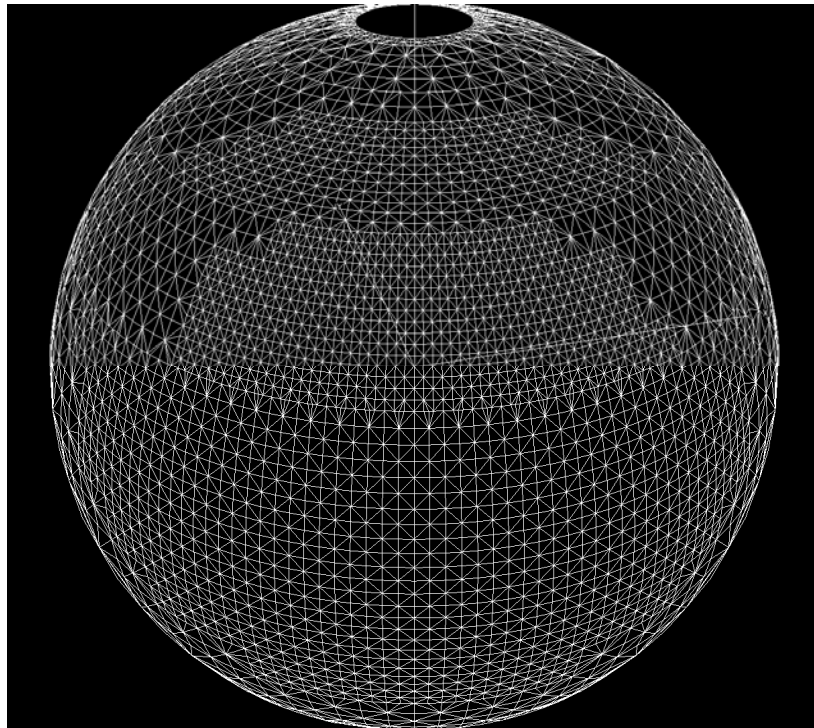


Figure 33 - *World* Object Using Node Area Calculation

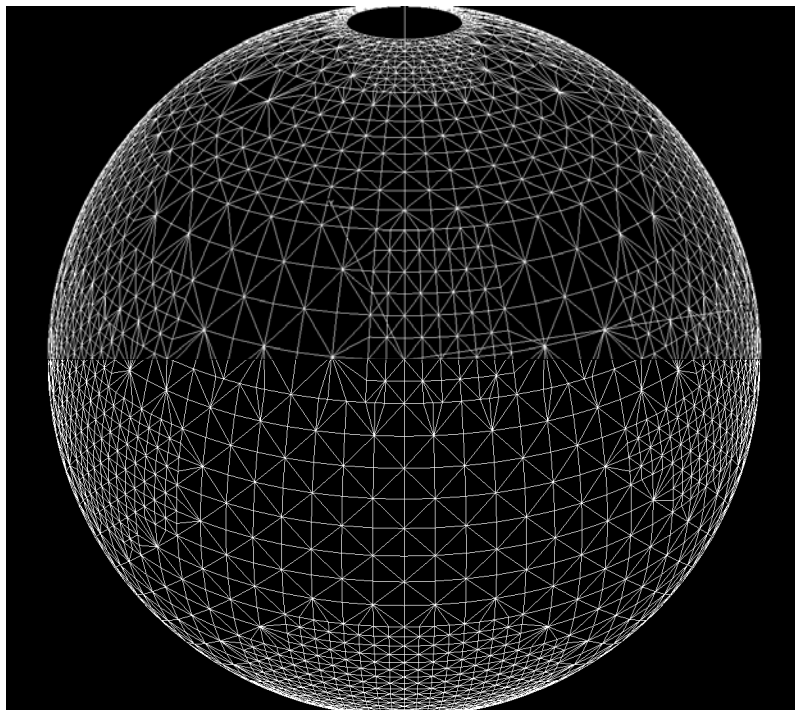


Figure 34 - *World* Object Using Screen Error Metric Calculation

The *Screen Error Metric* calculates the number of pixels that a line will shift when additional vertices are added by the creation of children. In Figure 35, the red vertex represents the new point that is added and the red line is the distance in pixels on the screen that the original line is shifted.

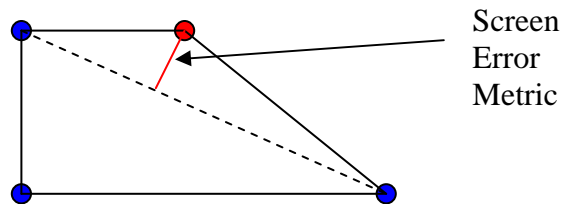


Figure 35 - Screen Error Metric

QuadTree Refinement Step - In order to speed up panning, the QuadTree is limited to increasing its depth by only one level per node per draw. This is accomplished by using a separate thread that is notified each time the QuadTree is updated if it was still possible to create more levels. This additional thread in turn tells JView to update the *World* object; this process repeats until there are no more possible levels to be created in the QuadTree.

Vertex Pool – The basic vertex object contained in the link list is a QTVertex. This object type is requested over and over again so a pool is created that vertices are drawn from, and returned to when no longer needed. This limits the number of times new QTVertex objects have to be allocated.

Texture Mapping – The *World* Object contains a list of textures; each center node contains a sub list of the textures that it contains from the master list. The node is drawn once for each texture on the sub texture list. This results in a multi-pass draw step that allows high-resolution imagery to be overlaid on low-resolution imagery. The TIROS satellite imagery and NASA's Blue Marble database are used as the default textures. Figure 36 shows the lowest resolution image from the Blue Marble database on the *World* Object.



Figure 36 - NASA's Blue Marble Lowest Resolution Image on *World* Object

Projections – In order to support the requirement for multiple projections, a generic *Projection* Interface was created along with two instantiations of that interface, *WGS84Projection* and *FlatProjection*. There are four methods required by the *Projection* Interface as shown in Table 1.

Table 1 - Method for Projection Interface

Method	Return Type	Parameter List
lonLatToXYZ	Vector3d	double lon double lat
lonLatToXYZ	Vector3d	double lon double lat double alt
lonLatToXYZ	Void	Vector3d _vert double lon double lat double alt
xyzToLonLat	Vector2d	double _x double _y double _z

The projection must be defined during *World* object creation by passing in the desired projection through the constructor. The default projection for the *World* object is WGS84 if no projection is defined during object creation. Screen shots of the *World* object for both WGS84 and Flat Earth Projections are shown in Figure 37 and Figure 38.



Figure 37 - *World* View with WGS84 Projection



Figure 38 - *World* View with Flat Projection

Dynamic Loading of Digital Terrain Elevation Data (DTED) – The *World* object has an option to use altitude that is enabled and disabled by a method call. If the *UseAltitude* option is set to true then DTED data is used to determine the altitude of all vertices contained in the QuadTree data structure. Up to level 2 DTED is supported if available on the user's hard disk, but interpolation will be used to estimate the elevation when only low-resolution data is available.

The QTVertex object stores the elevation for each vertex, and retains that information as long as the vertex is in the viewing frustum. When a vertex leaves the viewing frustum, the QTVertex object is de-referenced and becomes available for garbage collection.

Figure 39 and Figure 40 display screen shots of the *World* object with the use altitude option enabled.

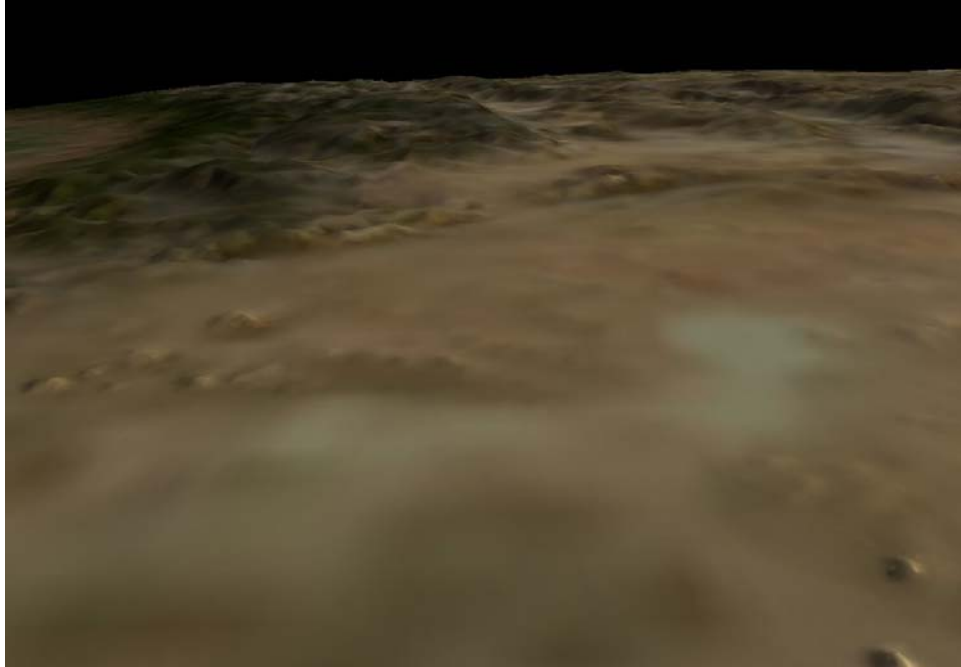


Figure 39 - DTED Enabled on *World* Object

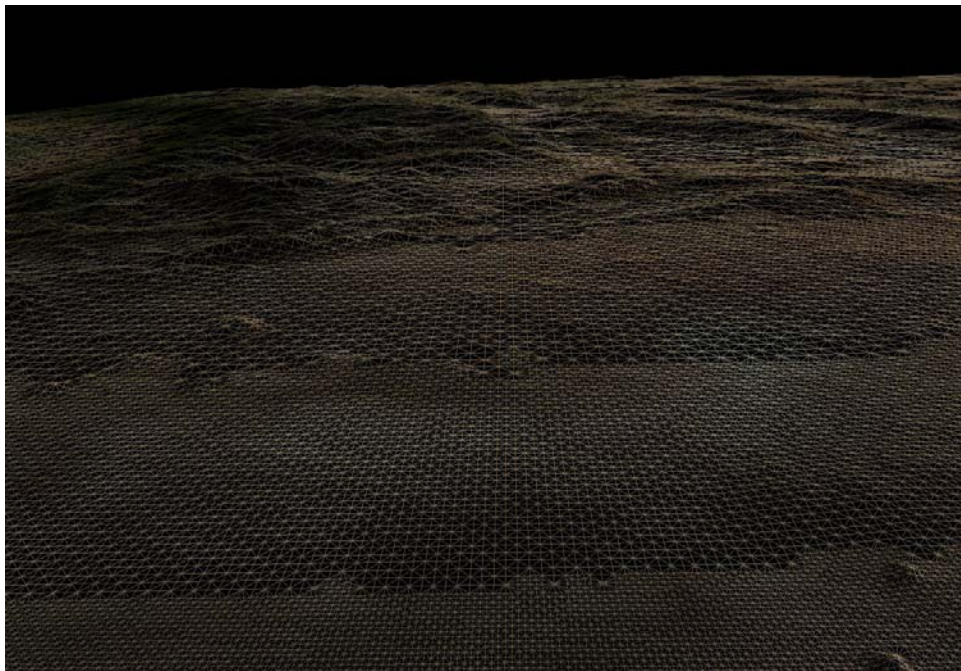


Figure 40 - DTED Enabled on *World* Object (Wire Frame)

Imagery Level of Detail (ILOD) – The *World* supports multiple levels of dynamically loaded imagery. A TextureElement object was created to represent an individual texture on the surface

of the *World*. The TextureElement object stores the bounds of the texture in degrees and is also responsible for loading the texture off the hard disk and onto the video card, as well as freeing the memory used when the texture is no longer needed.

The *World* object stores a JAVA ArrayList of all possible textures in order of image resolution. If a region of the *World* is represented by more than one level of detail, the texture with the lowest index on the list represents the lowest resolution image. Each QuadTreeNode object contains an additional list of textures for the sub set of textures that overlap that node. During creation of the QuadTree, the individual nodes create their sub textures lists by testing their boundaries against the boundaries of all textures on their parent's list. This has the effect of decreasing the number of possible textures a node might use at draw time since there might easily be thousands of textures for the entire *World*.

At draw time, each QuadTreeNode determines which textures are currently to be used by computing the Screen Pixel to Texture Texel Ratio (PTR) for each texture in their sub list. Calculation of the PTR requires several steps as shown in Equation 1, Equation 2, and Equation 3. The node creates an additional draw list of textures that contains only the textures that meet the PTR requirements.

$$PpD = \frac{NodeWidthPixels}{NodeWidthDegrees}$$

Equation 1 - Width of Node (Screen Pixels) / (Degrees)

$$DpT = \frac{WestLon - EastLon}{ImageWidthTexels}$$

Equation 2 - Width of Texture (Degrees) / (Texel)

$$PTR = \frac{PpD}{DpT}$$

Equation 3 - Screen Pixels per Imagery Texel Ratio

Dynamic Loading of Imagery, Texture Thread – All possible TextureElements are created at object creation but the actual imagery behind them is loaded from the hard disk or from a URL as needed by a separate *Texture Thread*. Each TextureElement has a *visible* and *loaded* flag that determine the state of the image. When a QuadTreeNode determines at draw time that it needs a texture, it sets the *visible* flag to true and if the texture has not been loaded, it adds a request to load that image to the Texture Thread. Each time the Texture Thread loads a texture, a Graph3D repaint is called. At the end of each draw step, the *World* object unloads all textures that it has in memory that are no longer visible. The Texture Thread will abandon any load texture requests

that have become not visible by the time they are processed. This will occur when the user pans quickly across the globe.

Multi-pass Draw Step – Each QuadTreeNode may have several textures that fall within its bounds and passed the PTR ratio test. This requires a draw pass for each texture contained in the nodes draw list. If any texture completely covers the node, no textures of lower resolution will be drawn. Figure 93 shows an example of a node indicated by 9 red circles and 4 different textures A, B, C, and D that all intersect with the node. If texture D is the highest resolution image then A, B, and C are skipped as they would be hidden by texture D. Textures are drawn in order of lowest resolution first since the last texture draw will be the one visible. If the reverse is true with A, B, and C at a higher resolution than D, all four will be drawn. Instead only individual textures that completely cover a node are detected.

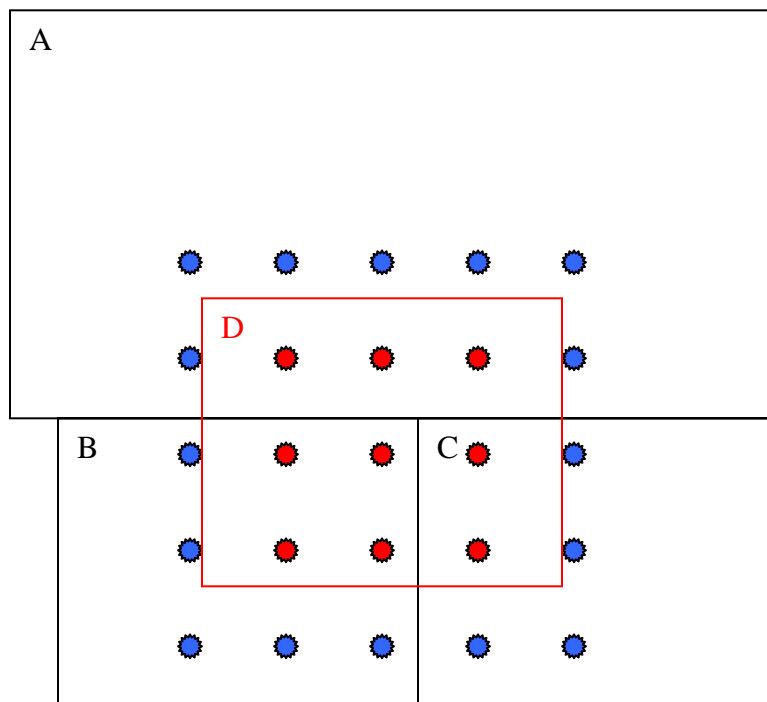


Figure 41 - Multi Pass Textures

The *World* does not require textures to end exactly on post boundaries which allows complete freedom in the use of input imagery. Texture D in Figure 41 demonstrates that showing textures coordinates have to be set for the blue posts as well as the red posts even though they are beyond the boundaries of the node. If texture coordinates were only given for the red posts, then the region of the texture between the red and blue posts would not be visible. Since OpenGL will interpolate the color of the edge texel in a texture for regions beyond the texture edge, it is necessary to surround all textures with a single texel wide transparent border. This prevents OpenGL from repeating the edge of each texture beyond its boundaries.

4.3.3 Mercator Projection Imagery Support

During this contract, CACI added support for imagery using the Mercator Projection. Several different types of imagery are available in the GIS Community that have been saved in the Mercator Projection. Mercator is a projection that decreases the size of imagery as latitude increases toward the north or south pole in an attempt to create landmasses that are correctly sized in latitude. For example, Greenland in a standard projection on a flat map looks as big as Africa. Figure 42 shows an example of an equally tiled map in Mercator displayed in WGS84 degrees. While the tiles are different sizes in WGS84, they would represent the same area in Mercator Projection. A *TextureSet* object was created *TextSetMercG* to support this type of imagery. At this time, the projection only supports the input of Mercator imagery and does not provide an interface to map the *JView World's* actual geometry. This means that the globe will still be drawn using WGS84 or Simple Cylindrical but the Mercator Imagery will be properly placed on the *World*.

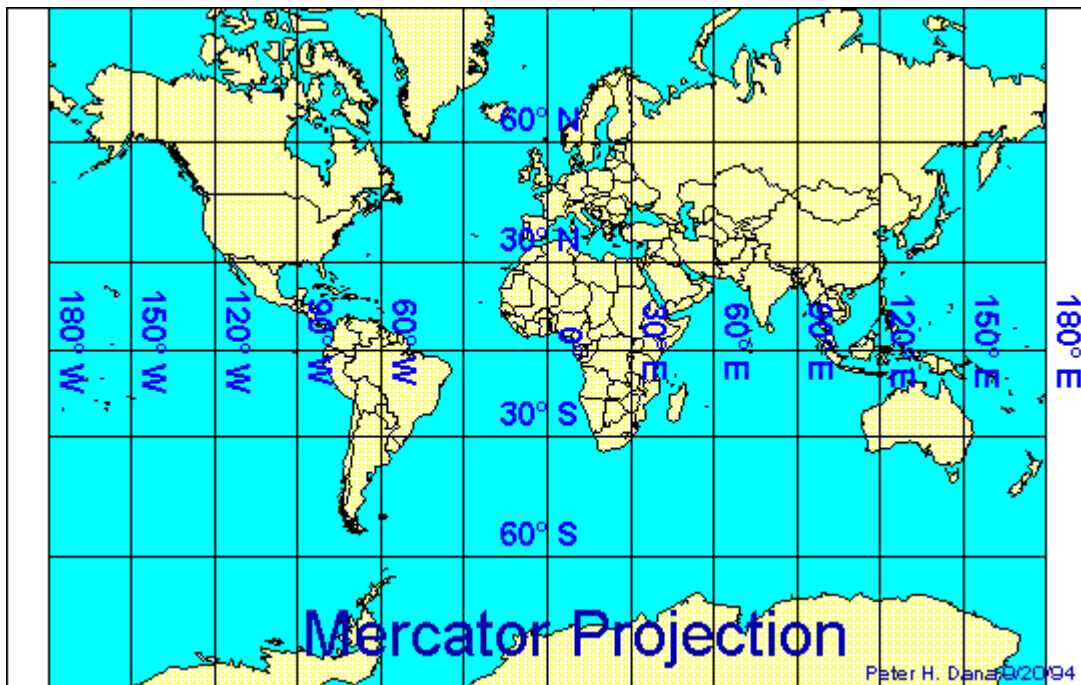


Figure 42 - Mercator Projection Coordinate

4.3.4 Lambert Conic Projection

CACI implemented and tested a Lambert conic projection for use by the *JView World*. This projection is only used for mapping the *World's* vertices and has nothing to do with the type of imagery displayed. The Lambert projection insures correct latitude image sizing for map data within a small region bounded by two lines of latitude. This projection would only be desirable if the area of interest was relatively small like a US state. In Figure 43, the left side shows the

Lambert conic projection of the *World* with its latitude cuts at 50 and 40 degrees north and the central meridian at 0. In this example, the *World* will preserve the map data for the region 40 degrees to 50 degrees north latitude.

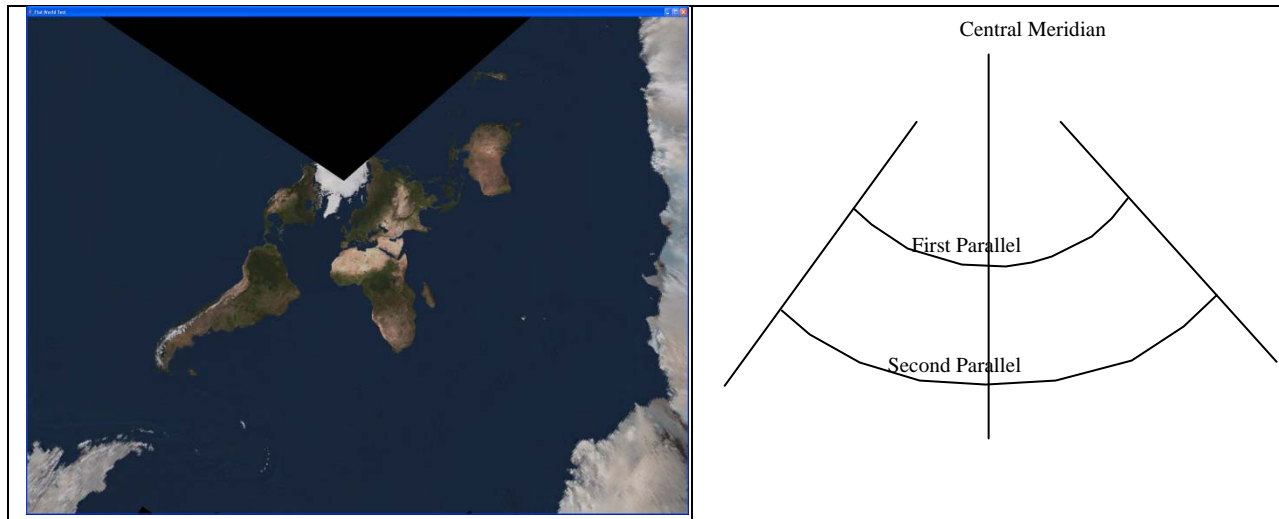


Figure 43 - Lambert Conic Projection

4.3.5 Raster Product Format (RPF) and Compressed ARC Digitized Raster Graphics (CADRG) Data Support

During this contract, the ability to use RPF and CADRG data directly off the source CDs was added to the *World* object. RPF/CADRG data comes in many different types and levels of detail. The user defines what type of data to display and the object manages which data is currently visible and at what distance to use it. Figure 44 displays Controlled Image Base (CIB) 10 meter data displayed on DTED level 1 of the Trinity Site in Southern New Mexico, the site of the first Atomic Bomb test.

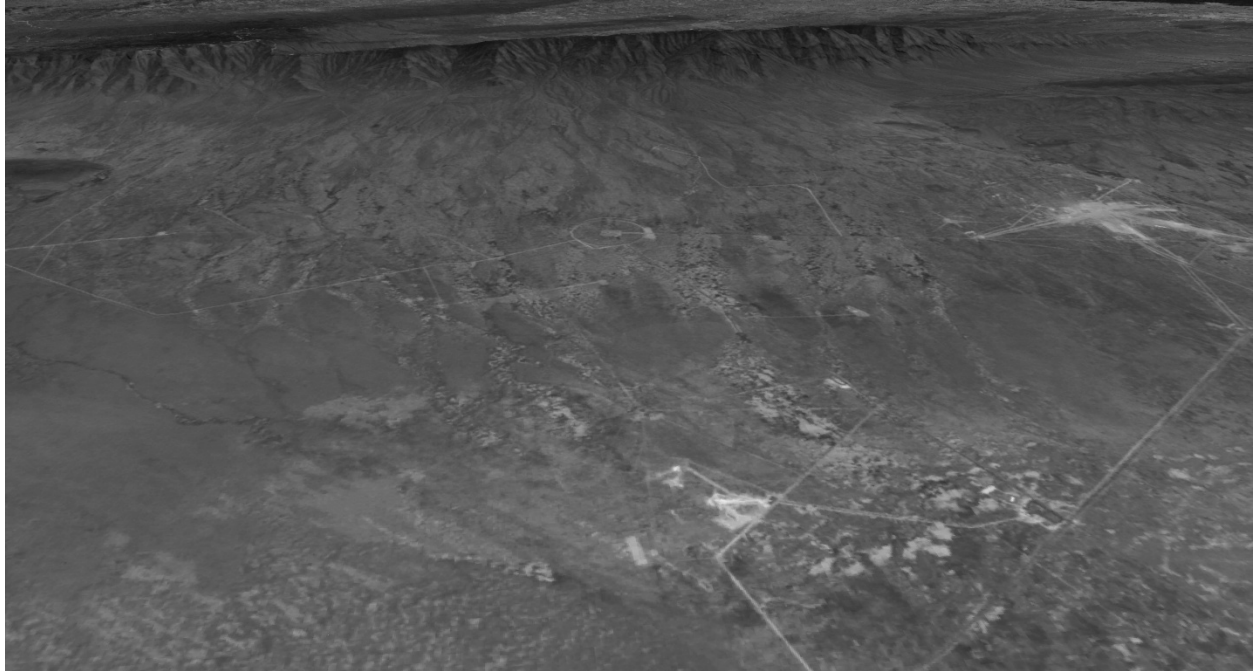


Figure 44 - Trinity Site with CIB 10 meter

4.3.6 TerraServer-USA Data

TerraServer-USA is an internet site that serves United States Geological Survey (USGS) public domain imagery of the United States. The imagery comes in three types which are as follows: Digital Ortho Quadrangle (DOQ) aerial, Digital Raster Graphic (DRG) topographical, and URBAN high resolution natural color ortho-imagery. Each type comes in multiple resolutions ranging from .25 meters per pixel to 512 meters per pixel. All USGS data is projected into the Universe Transverse Mercator (UTM) projection using the North American Datum of 1983 (NAD83). Figure 45 shows all valid types, resolution and the available UTM zones.

Theme	Scale Value	Resolution (Meters Per Pixel)	UTM Multiplier
Urban	8	.25	50
Urban	9	.5	100
DOQ, Urban	10	1	200
DOQ, DRG, Urban	11	2	400
DOQ, DRG, Urban	12	4	800
DOQ, DRG, Urban	13	8	1600
DOQ, DRG, Urban	14	16	3200
DOQ, DRG, Urban	15	32	6400
DOQ, DRG, Urban	16	64	12800
DOQ, DRG, Urban	17	128	25600
DOQ, DRG, Urban	18	256	51200
DOQ, DRG, Urban	19	512	102400

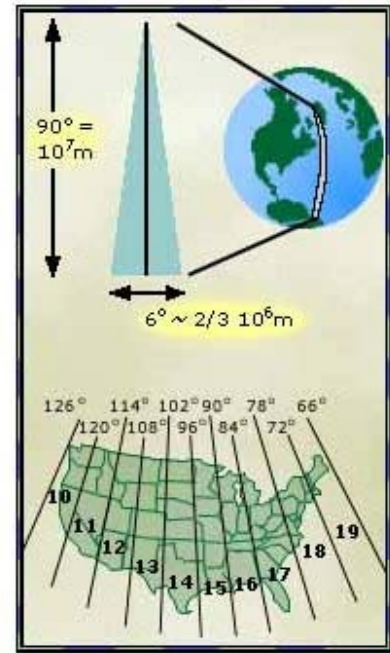


Figure 45 - TerraServer-USA Types and Resolutions

TerraServer tiles are requested using theme, scale, resolution, and tile identifiers X and Y. The X and Y tile identifiers are calculated from the UTM coordinates by dividing the UTM easting for X, and northing for Y by the UTM Multiplier.

Each downloaded image tile is a 200x200 pixel JPEG and since OpenGL requires powers of two for use as a texture, each tile is copied into a 256x256 OpenGL byte array with the borders padded with transparent pixels before it is copied onto the video card.

During peak times of the day, TerraServer-USA can be very slow to serve data and in the initial implementation this caused the texture thread to hang while waiting for TerraServer to respond. To solve this problem, an additional thread was created to process only URL requests thus preventing the primary texture thread from being delayed.

Figure 46 shows a screen shot of the *World* displaying TerraServer-USA URBAN imagery over Albuquerque, NM. The white area in the upper right corner of the imagery represents an area where only partial imagery is available and since JPEG images do not support transparency, pure white is used instead.



Figure 46 - TerraServer-USA URBAN Imagery

4.3.7 Create DTEDImageGenerator TextureSet Object

A TextureSet was created to allow the *World* to automatically generate imagery using the DTEDImageGenerator developed under this contract. This texture set caches all generated imagery so that repeated requests will result in much faster load times.

The DTEDImageGenerator creates a color relief map from DTED data by mapping elevations to an array of user defined colors that are blended to create a range of possible colors. These generated images do an excellent job of representing change in elevation, but depending on the color, may not create very realistic looking imagery. Figure 47 shows an image created of New Mexico using the default color array.

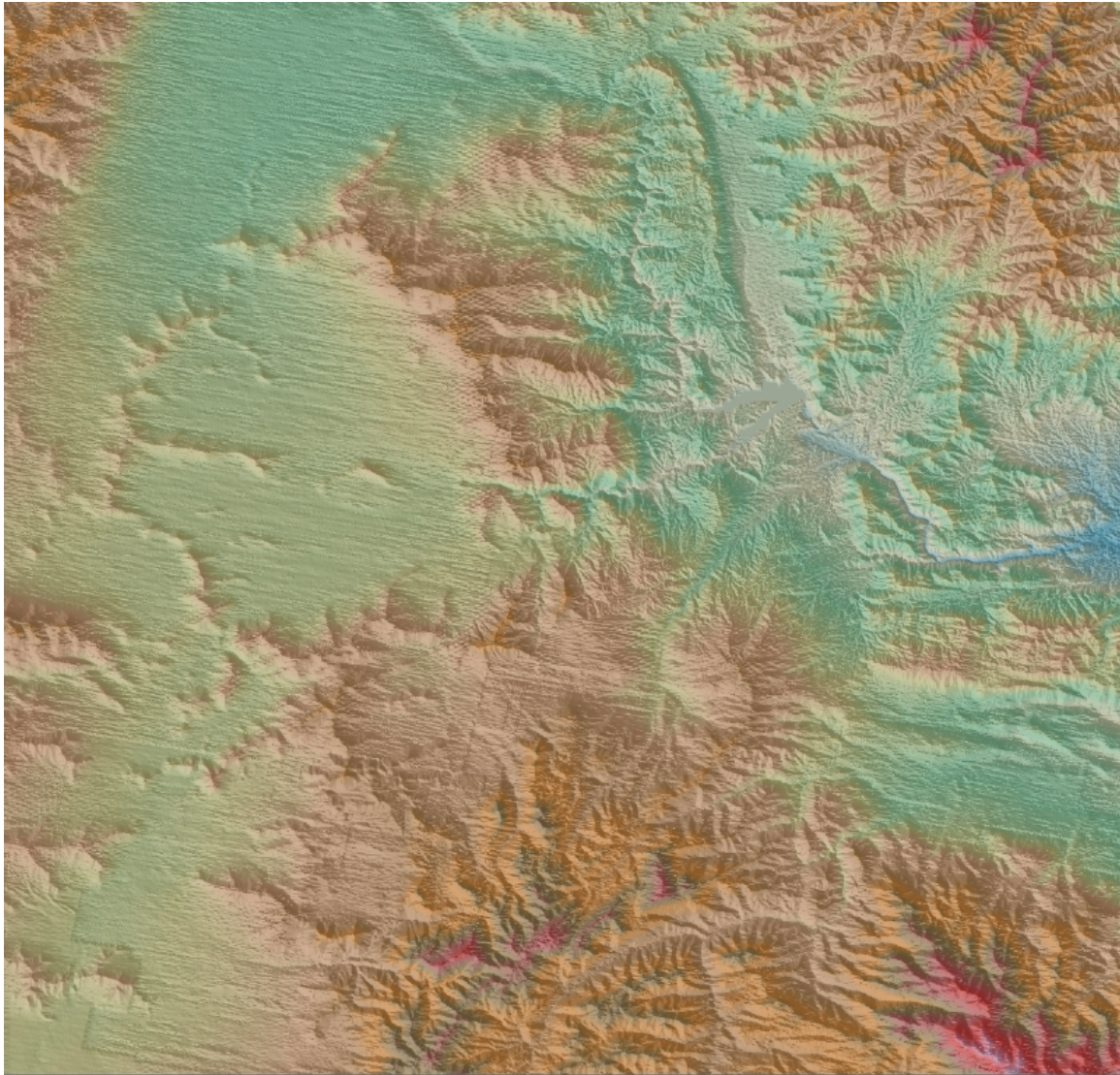


Figure 47 - DTEDImageGenerator Original Imagery

In order to create more detailed color maps, we created a utility that could sample imagery and then compare the DTED elevations for each pixel in the source imagery and automatically create color array data. The intended purpose was to use satellite photos or NASA generated imagery to create a color map for use by the DTEDImageGenerator that would allow it to create more realistic imagery. Figure 48 shows a source image of North America generated by NASA using unknown methods and Figure 49 shows a DTEDImageGenerator image created using the color map sampled from the NASA image.



Figure 48 - NASA Generated Image of North America

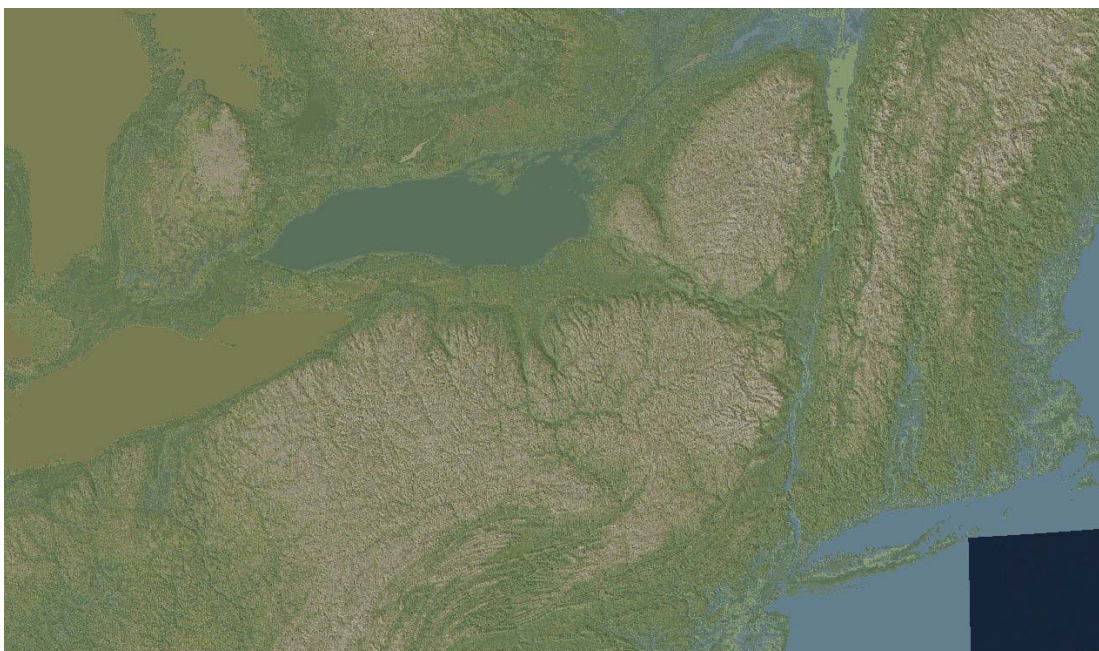


Figure 49 - DTEDImageGenerator Image using NASA Color Map

Since these sample color maps could be thousands of colors in length, a more realistic, and manageable method of defining them is provided. The color sampling utility outputs the color data into a XML file format and the DTEDImageGenerator parses the XML files to define all possible user configurable settings. Figure 50 shows a sample XML input file for the DTEDImageGenerator with the colors entry truncated to make the example a reasonable length.

```
<DTED_IMAGE_GENERATOR>
```

```
  <SETTINGS
```

```
    configName="config_1"
```

```
    minAlt="5"
```

```
    maxAlt="8000"
```

```
    autoAdjustColorBins="false"
```

```
    grayScale="true"
```

```
    grayBlendPercent="1"
```

```
    fill="true"
```

```
    fillBlendPercent=".4"
```

```
    interpolate="true"
```

```
    contours="false"
```

```
    findLakes="false"
```

```
    findLakesCutOff="300"
```

```
  />
```

```
<!-- Elevation R G B A -->
```

```
  <COLORS>
```

```
    0 59 129 163 255
```

```
    1 60 129 162 255
```

```
    2 41 96 65 255
```

```
    ...
```

```
    ... truncated
```

```
    ...
```

```
    1891 167 162 104 255
```

```
    2132 71 81 28 255
```

```
  </COLORS>
```

```
</DTED_IMAGE_GENERATOR>
```

Figure 50 - XML Input File Example for DTEDImageGenerator

Figure 51 shows an example of this imagery displayed on the *World* object.

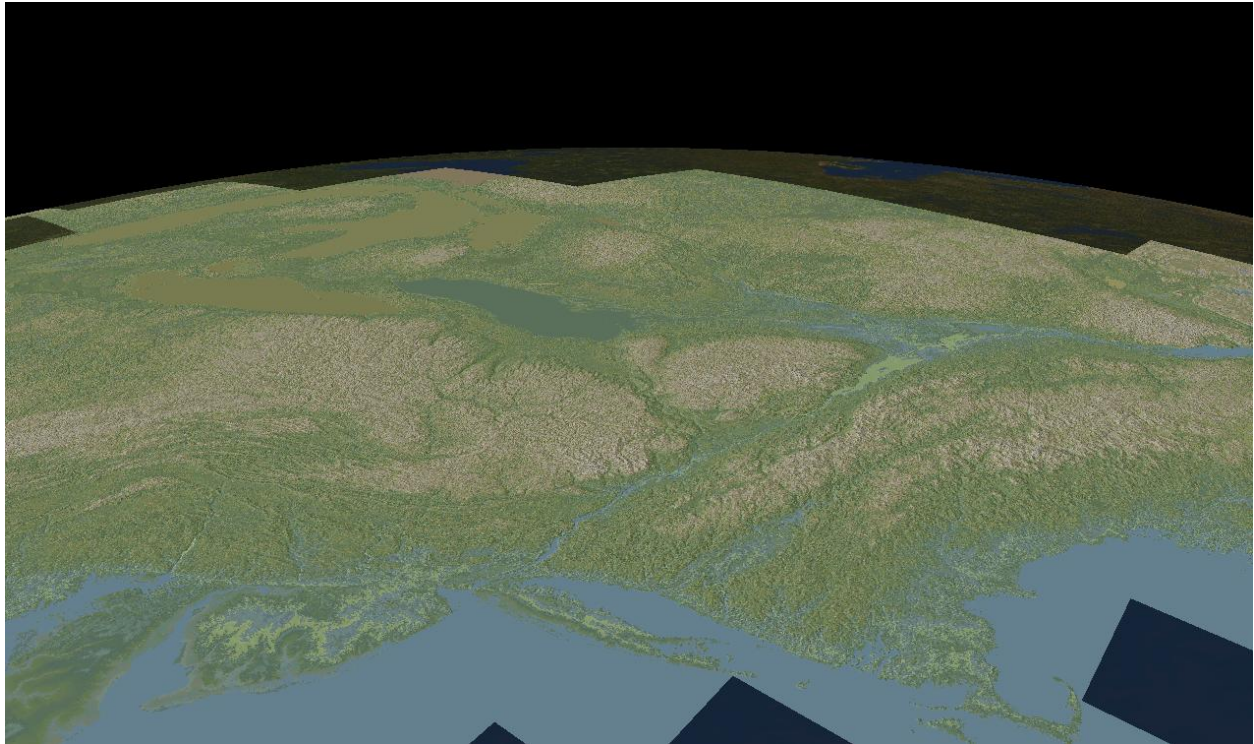


Figure 51 - DTEDImageGenerator DTED Level 0 Image displayed on *World*

4.3.8 NASA World Wind Imagery Support

NASA has recently ported the *World Wind* Application to a Java based SDK [2]. This provides developers access to NASA database of imagery over the internet. CACI added support to stream this imagery for display on the JView *World* and to store NASA's imagery in a local cache. When accessing the same region cache, data will be loaded first reducing download requirements. Cache structure is compatible with NASA cache and may be shared between the JView and NASA API's. Figure 52 shows an example of JView *World* with NASA's *World Wind* imagery over Albuquerque, New Mexico.



Figure 52 - NASA *WorldWind* Imagery of Albuquerque, New Mexico

4.3.9 Web Map Service (WMS) Imagery Support

Web Map Service servers are freely available internet servers that produce maps of spatially referenced data dynamically from disparate geographic information. Servers are available with terrain imagery, political boundaries, cities, roads, rivers, instant weather, and many other options. During this contract, CACI added support to stream this imagery for display on the *JView World* and to store imagery in a local cache. Many of the WMS servers are very slow but the cache mechanism makes it possible to load imagery off-line and then bypass the slow server. Figure 53 shows an example of the northeast US with political borders, major cities, and instantaneous weather data.

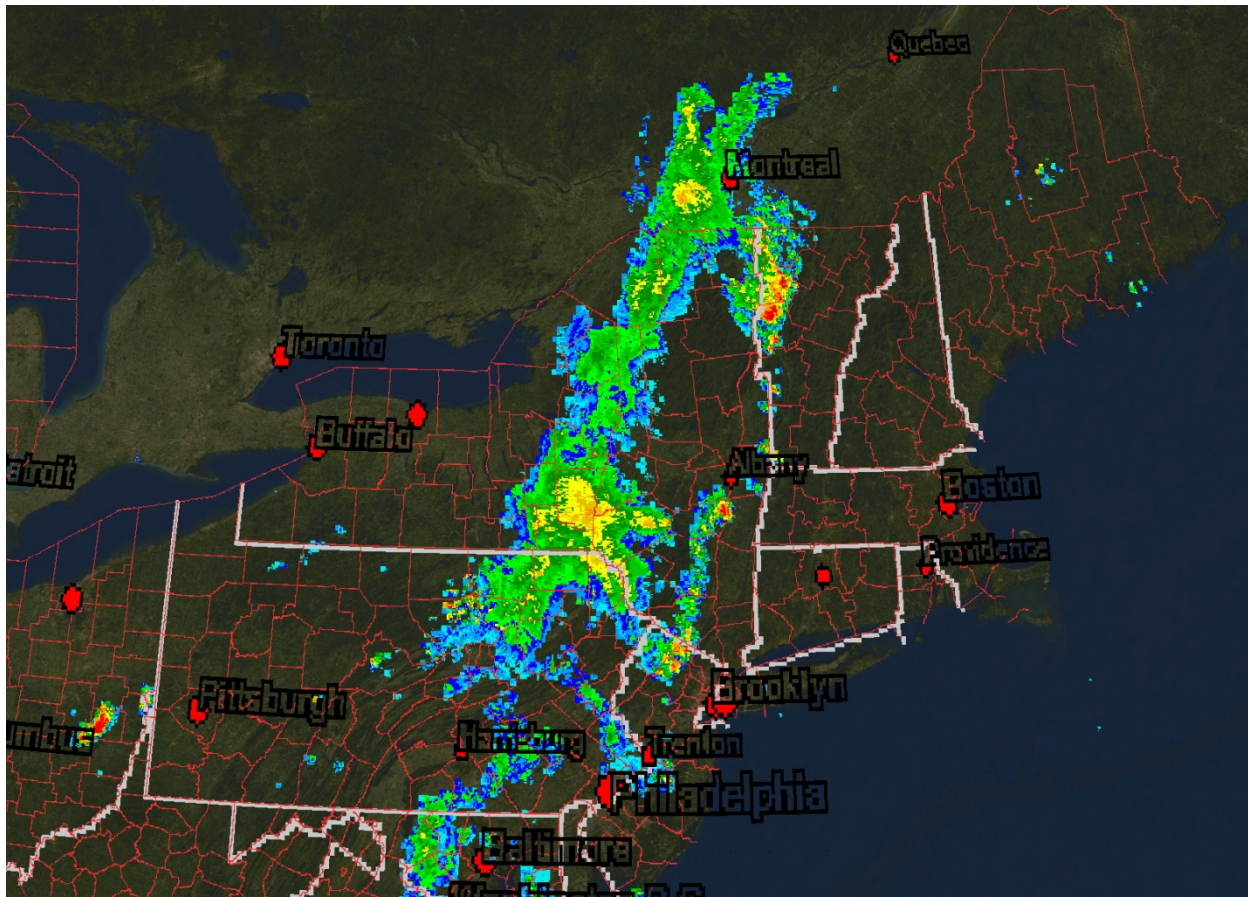


Figure 53 - WMS Image of Current Weather over NE US

4.3.10 Image Cache Object

A basic Image Cache Object was created (initially to supplement TerraServer data) that stores downloaded imagery on the local system. The *World* checks the cache object before attempting to download imagery from the TerraServer site and to write the image out to the cache after new imagery is downloaded.

4.3.11 Statistics Generation

As a debugging tool for determining how well the JView *World's* texture processing engine is working, several statistics are internally tracked and can be accessed through the `fillWorldStats` method. Some of the statistics that are tracked are shown in Figure 54.

Texture Statistic	Meaning
Loaded on Card	Number of textures that are currently stored on the Video Card
Ever Loaded	Total number of textures that have ever been loaded onto the Video Card since the JView <i>World</i> was allocated
Ever Unloaded	Total number of texture that have ever been unloaded from the Video Card since the JView <i>World</i> was allocated
With No Data	When the <i>World</i> requests a data type that is not contained on the hard disk but has been enabled by the user a <i>No Data</i> Texture Element is created, this statistic counts the number of those elements currently in memory.
Rejected Due to Visibility	Number of textures that were added to the texture queue, but were no longer visible at the time the queue was ready to process the texture request. Primarily caused by panning away from texture area before data is loaded.
Rejected by Queue	Number of textures rejected by queue, this represents an error and should always be zero.

Figure 54 - Texture Specific Statistics

4.3.12 Texture Reduction

CACI implemented an option that uses a nodes' distance from the camera to rapidly reduce the resolution of textures drawn as distance from the camera increases. This option calculates the distance to the closest node during the update geometry step and then uses the ratio of the current node distance from the camera to the closest node distance to drop off textures distant from the camera focus point. This option drastically decreases the number of textures loaded in areas that have multiple levels of imagery such as the Albuquerque Terra Server data. For a test location above Albuquerque, NM, with the new option disabled, the *World* loads 772 textures for a total of 200 Mb on the video card, with the *limit textures* option enabled, this drops to 211 textures and 60.5 Mb of texture data on the video card. Since the Albuquerque area has 19 levels of imagery resolution available, this still creates an attractive scene with only nodes far from the camera showing a drastic reduction in imagery quality. Since most machines do not possess 200Mb of video memory, this option is a very attractive for this region and data type. Figure 55 and Figure 56 show the differences for the Albuquerque, NM test.

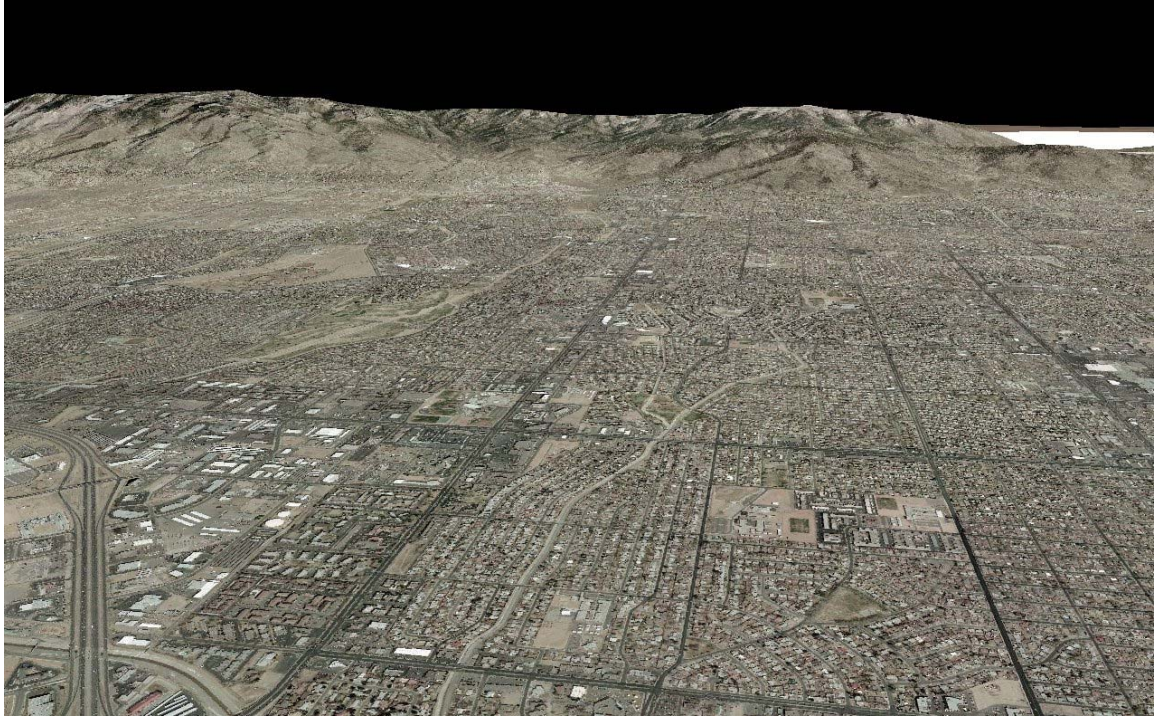


Figure 55 - Albuquerque, NM Terra Server Data Not Limited (772 textures and 200Mb)

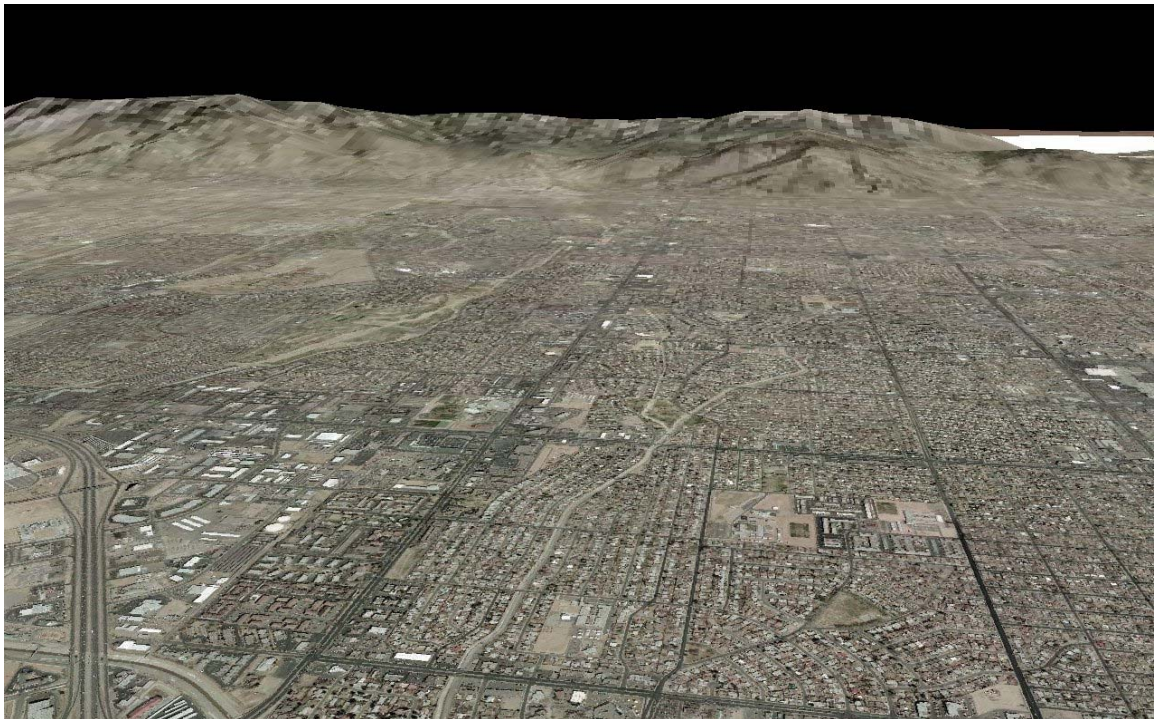


Figure 56 - Albuquerque, NM Terra Server Data Limited (211 textures and 60.5 Mb)

4.3.13 WorldSettings Object

The *World* contains 28 user configurable parameters. Using the standard object oriented design methodology, this requires 28 object methods for assigning and retrieving their values. In order to simplify this process and eliminate errors resulting from the order of operations, the *World* uses a single *WorldSettings* objects for all user configurable settings. This object may be modified as the users' needs require and when the settings object is passed back to the *World*, a copy is saved and applied automatically. This retains encapsulation between the *World* internal setting and the user.

4.3.14 Multi-View Support

CACI did a large amount of work to allow the *World* to support multiple simultaneous views. There are three major steps required to allow the *World* to properly support multiple views:

- No static *World* member variables that would cause separate instants to interfere with each other.
- Share DTED post data with all views.
- Share Texture data with all views.

In order to eliminate the use of static *World* members all threads accessing *World* member data now do so through a *World* reference instead of through the class name. This required many objects to have a member added to store a reference to the *World* it belongs to. This change made it possible for multiple Graph3Ds each with their own *World* object to be created. It is not, and will not be possible to add a single *World* object to multiple Graph3Ds. A new instant of the *World* must be created for each Graph3D.

Sharing DTED post data with all views was a significant design that created the framework by which the texture data can be shared in the future. A new object was created to store all data specific to a single DTED post that is not view dependent. All view or *World* dependent information is still stored in a QTVertex object, which in turn has a reference to a DTED post data object. Each *World* contains its own queue of DTED posts to be processed; the main DTED thread now calls on each *World* object to process its own DTED queue.

4.3.15 Relative Altitude Look Up

The *World* contains a method *getAltRelative* used to determine the altitude of a point based on the current triangulation of the *World*. This is necessary to attach elements to the surface of the *World* such as tank models. Once the deepest current created node containing the requested point is identified, the altitude is interpolated using the 4 vertices that bound the point. The accuracy of this value, as compared to the actual altitude of the point increases as the camera zooms in, or as the *World's* node ratio is increased.

4.3.16 Grounding Elements

When attaching objects to the terrain *Grounding* is always difficult and never precisely correct. Each object might need to be grounded differently from other objects. CACI has implemented a package of grounding policies that can be applied to objects drawn on the surface of the JView *World*.

4.3.16.1 Grounding Policy

CACI has developed six different grounding policies that a user can choose from depending on how they want an object to be displayed on the terrain surface. These six grounding policies are as follows:

1. Touch-Ground
2. In-Ground
3. Center-To-Ground
4. Percent-In-Ground
5. Parallel-To-Ground
6. Average-Parallel-To-Ground

The Touch-Ground policy will take an element and ground it so that the bottom most vertices of the geometry just touch the ground. This is done by first traversing through the nodes of the geometry and grabbing the set of vertices that are the lowest. Then the highest elevation is calculated from each location of the lowest vertices. Touch-Ground policy will return a coordinate that has the original latitude and longitude with a new elevation. Figure 57 shows an object that is using the Touch-Ground policy. It can be seen that the object is only touching the ground on the side of the hill. The In-Ground policy does the reverse of Touch-Ground policy. The bottom most vertices at least touch or are below the ground. Figure 58 shows the same object with the In-Ground policy.

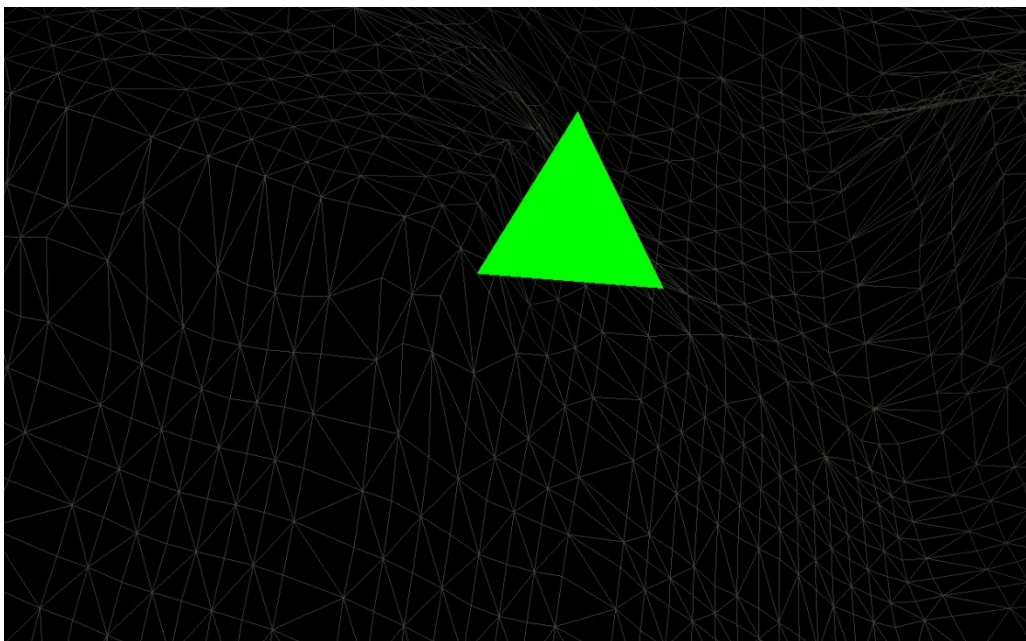


Figure 57 - Touch-Ground policy

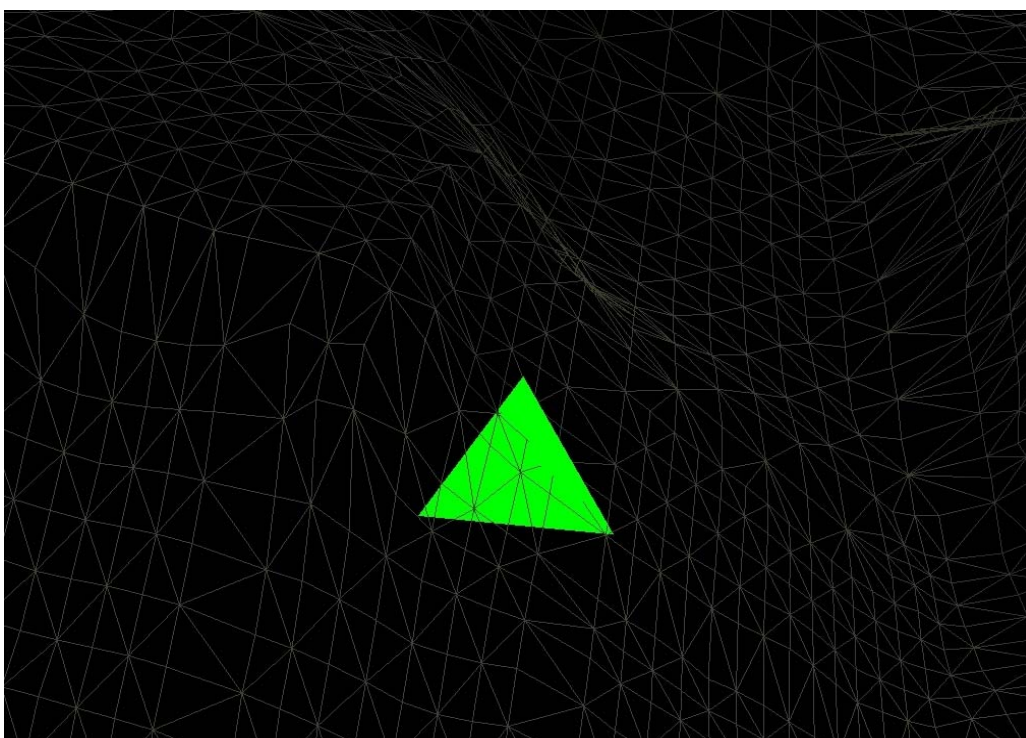


Figure 58 - In-Ground policy

Center-To-Ground policy sets only the center point of the model to be at ground level. Since this policy does not examine the element vertices to determine high and low points it is computationally much less expensive than the previous two.

Percent-In-Ground policy uses a defined percentage between the Touch-Ground and In-Ground policies. This policy is useful for when an element needs to be grounded a little deeper than Touch-Ground but not as deep as In-Ground.

Parallel-To-Ground is an enhanced version of Touch-Ground that includes modifying the rotations of the object so that it sits in the same plane as the terrain triangle it is grounded to. The calculations of finding the resulting orientation are in WGS rotation which is a different axis of rotation than the JView's axis of rotations. Figure 59 shows the differences between these two rotations.

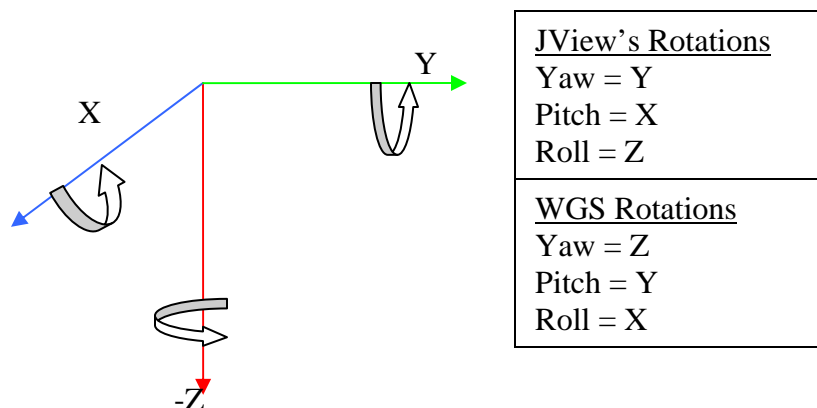


Figure 59 - Orientation definitions

Average-Parallel-To-Ground policy modifies the Parallel-To-Ground policy by calculating the average ground height of the lowest vertices contained in the model and then grounds the object to that elevation. As with Parallel-To-Ground, the orientation of the object is modified to place the model in the same plane as the terrain triangle.

4.3.17 *World* Automated Test Program (WATP)

CACI developed an automated test program used in comparing JView *World* against a set of test parameters. We often find that changes to one part of the JView *World* result in unexpected effects in other areas. The creation and use of the ATP, reduced the time lost tracking the source of many bugs by catching them immediately after they are introduced. Figure 60 shows a sample test used to compare a go to command to a previously stored result for that operation. Any change in the test parameters would require a programmer look at these results and determine if the changes were correct.

```

TEST:: DAM (CA) PASSED
-----
TEST:: Trinity PASSED
-----
TEST:: Whitesands --- FAILED --NODES DRAWN ARE NOT EQUAL
*****STANDARD RESULTS*****
NAME = TEST:: Whitesands
ANCHOR DISTANCE
    Longitude : 32.64510412717321
    Latitude  : -106.52555206083908
    Altitude  : 22.40498389022394
CAMERA DISTANCE
    Elevation : 90.0
    Azimuth   : -0.2528724054748504
    DISTANCE  : 1953692.1459967892

    NODE_DRAWN      6
    NODE_IN_TREE     338
    NUM_LOADED_TEXTURE 2
    TRIANGLES_DRAWN  59
    DEEPEST_LV      6

*****CURRENT RESULTS*****
NAME = TEST:: Whitesands
ANCHOR DISTANCE
    Longitude : 32.64510412717321
    Latitude  : -106.52555206083908
    Altitude  : 22.40498389022394
CAMERA DISTANCE
    Elevation : 90.0
    Azimuth   : -0.2528724054748504
    DISTANCE  : 1953692.1459967892

    NODE_DRAWN      260
    NODE_IN_TREE     894
    NUM_LOADED_TEXTURE 2
    TRIANGLES_DRAWN  2080
    DEEPEST_LV      11

```

Figure 60 - *World* Test Results

4.3.18 Node Change Listeners

Since the JView *World* continuously refines its geometry as the user pans and zooms, the actual geometry of the earth is moving up and down as more DTED data is read in. Any point on the surface of the earth not exactly on a DTED post is actually an estimate based on the triangles drawn for the current level of detail. Objects that are to be drawn on the surface such as buildings or automobiles creates a problem since it could result in the object either floating above or below the actual surface. CACI developed a NodeChangeListener to the *World* that allows users to register listeners that will notify an object that its ground location has changed altitude allowing its XYZ coordinate to be updated. Updating objects can be a very expensive CPU intensive operation requiring many trigonometry calculations.

4.3.19 Demo Browser – World Demo

CACI updated the demo browser to include demonstrations on how to create a simple JView *World*. We added two demonstrations, one that creates a basic *World* with textures, and a second that creates a basic *World* without textures and with DTED. Both demos come with a detailed description of what the demo represents and how to run the demo. Figure 61 shows the demos that were added to the demo browser. The upper two images represent the *World* with textures and the lower image represents the *World* without textures.

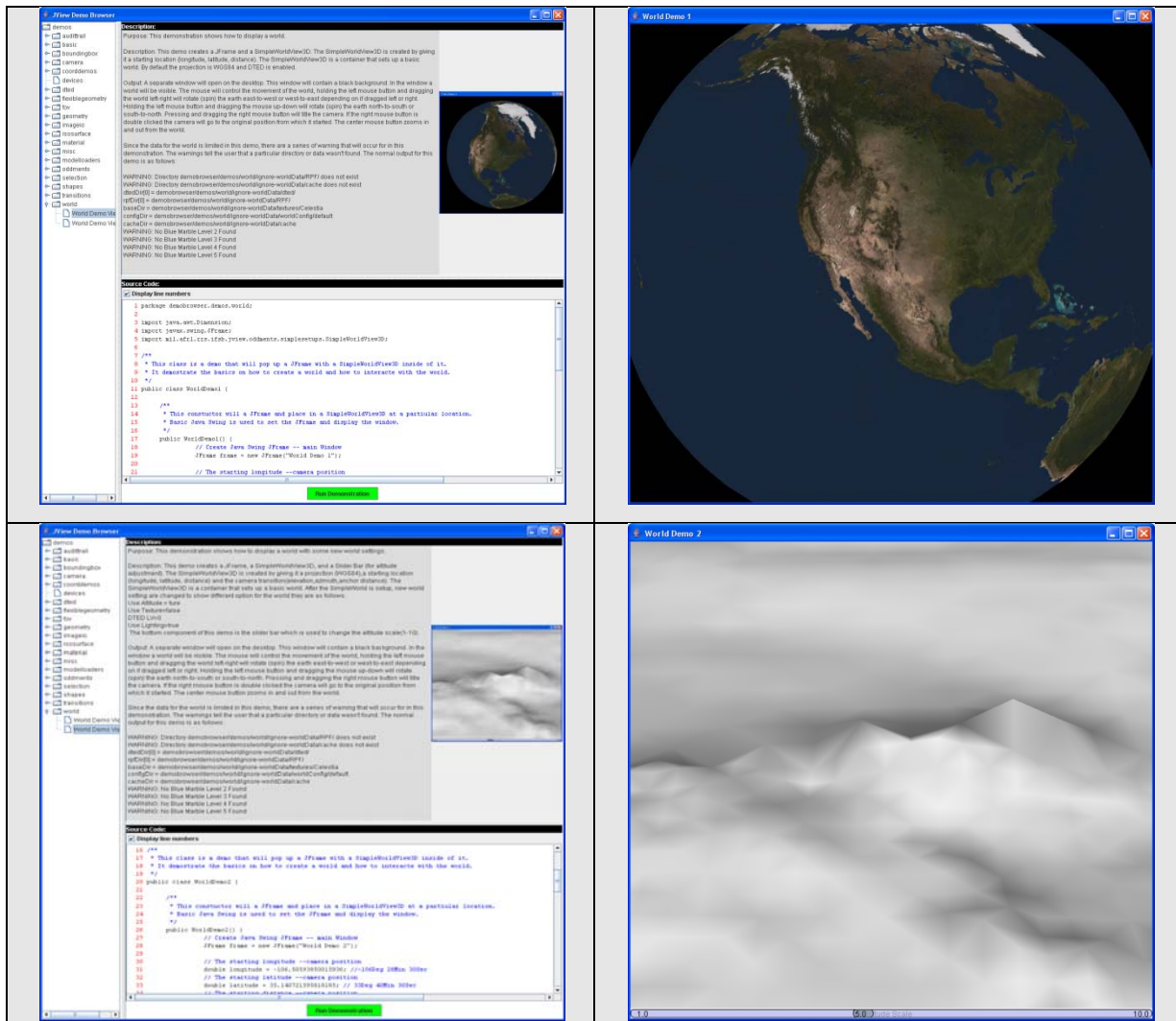


Figure 61 - Demo Browser – *World*

4.4 Developed the initial design framework for providing visualization support to the Virtual Airspace Modeling and Simulations Project (VAMS)

This portion of the VAMS effort aims to create an application to visualize data from various sources that will aid in the interpretation of the effects of simulated changes to the operation of the NAS, and potentially, to aid in the visualization of live data indicating the actual state of the air system. NASA is currently developing a complex model and simulation of the NAS under a separate VAMS project, the ACES simulation. ACES attempts to model all of the key components of the NAS with very fine granularity and generates vast amounts of data for each simulation run. ACES is presently under active development, therefore the types of data generated by the simulation are still growing. Additionally, ACES relies on a significant volume of input data to specify the parameters of the simulation, which is also growing as ACES evolves.

The ACES Viewer provides support for loading the data produced (and consumed) by ACES, and for processing and visually encoding the data for display. More specifically, the Viewer will provide functionality to assist in analyses of the simulation data by allowing users to construct visualizations that enhance the ability of analysts to elicit insight into the problem domain that would otherwise be difficult or impossible to obtain. To this end, the Viewer falls into a category of tools commonly referred to as an *exploratory, composable visualization system*. The system is composable in regard to the mapping and transformation of raw input data to an associated visual representation. Users will select the appropriate combination of input data, transformations, and visual atoms to create or compose a visualization. Exploratory refers to the fact that a user may not know what information they are looking for during analysis of the simulation run(s). The Viewer will allow users to interact with a particular visualization to explore the data in a number of different ways, including:

- Visual Querying, for example, drilling down to show more detailed information, or adding display of related information via direct interaction with the visualization.
- Tuning parameters of visual atoms to adjust the appearance of the visualizations.
- Transformation of input data, including aggregation (counting, averages, grouping, etc.), comparison (subtraction, evaluation of relationships, etc.), filtering, sorting, and others.
- Dynamically modifying the composition of a visualization.
- *Re-playing* the simulation, showing the temporal aspect of the data where appropriate.

The primary source of input to the ACES Viewer will be provided by the ACES simulation. Since ACES is still under active development, the Viewer must provide a relatively simple mechanism to add support for new data types and to modify support for existing data types. The ACES Viewer must also provide a mechanism by which users can access and visualize custom data produced by NAS researchers during analysis and evaluation process. Other input to the

Viewer will come in the form of user interaction with the application. This type of input includes:

- Specification of composition graphs defining the components and component relationships of each visualization.
- Interaction with the composed visualizations.

Output from the application consists of the visualizations generated by the users. The Viewer will also provide a mechanism for accessing the visualizations outside of the application through generation of static images and movies. This will allow the visualizations to be embedded within presentations or other media without necessitating availability of input data, or the complexity of using the ACES Viewer application. Finally, the composition graphs defining a visualization will also be an output artifact of the Viewer. Users will be able to save the visualization configuration to facilitate application of the composed display to alternate input (e.g. another simulation run), and sharing with other users/analysts.

4.4.1 Application Infrastructure

4.4.1.1 Visualization Components

This section provides a brief introduction to the components of a visualization in the ACES Viewer.

4.4.1.1.1 Data Tables

The ACES Viewer requires some data to visualize. This data may come from a variety of sources external to the application. Once imported, all data in the ACES Viewer is represented in tabular form, where a Table is an ordered collection of Tuples, and a Tuple is an ordered collection of data fields. All Tuples in a single Table are comprised of the same sequence of data fields (although their values will vary). Tables in the ACES Viewer are analogous to tables in a relational database system with Tuples corresponding to rows in the database table. Each Table in the ACES Viewer has an associated Schema that specifies the fields that must appear in each Tuple in the Table, including a field name, data type, and an optional data unit where supported by the data type.

Many types of data can be naturally represented in this form as evidenced by the ubiquity of relational database systems. The ACES Viewer directly supports data from relational database systems as well as Comma Separated Values (CSV) files. Additional data sources may be supported through plugins to the application.

Data Access Tables (Figure 62) in the ACES Viewer are differentiated from other Tables in the application (Transforms) by the fact that they do not accept input from other Tables. This statement will become clearer after the sections describing Transforms and Visualization Construction.

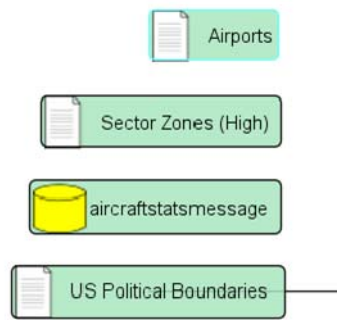


Figure 62 - Several Data Tables in the ACES Viewer Visualization Graph

4.4.1.1.2 Transforms

The ACES Viewer supports various types of data processing via Transform components. Transforms (Figure 63) can be used for a variety of purposes, including combining data from multiple input Tables, evaluating aggregate functions, analyzing data, annotating data, or marshaling data into alternate formats. Transforms are a special type of data Table, where the content (comprised of Tuples) is defined by the transformation function evaluated against one or more input tables. Transforms are differentiated from Data Tables in the ACES Viewer by the fact that they accept (and in fact require) input data in order to function.

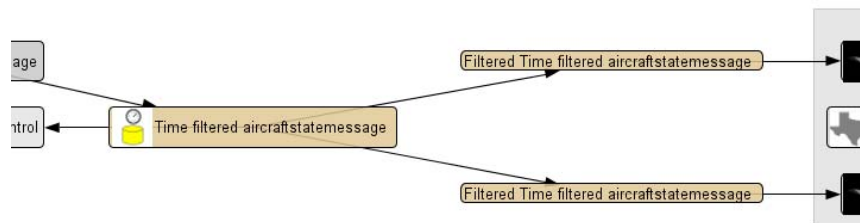


Figure 63 - Several Transforms included with the ACES Viewer

4.4.1.1.3 Display Containers

Display Containers represent the drawing surface to which visualizations will be rendered. In its present state, the ACES Viewer provides a 3D Scene as the sole Display Container. A Display Container can contain any number of Renderers that are responsible for drawing the visual representation of data in the Display Container. Note that since the Display Container effectively defines the visualization style (3D, plots, tables, etc.), only Renderers that support that particular style may be used with a particular Display Container type. Since there is only one Display Container type present in the application, all bundled Renderer types are compatible

with the 3D Scene, however ACES Viewer plugins may provide alternate Display Container and/or Renderer types.

4.4.1.1.4 Renderers

Renderers (Figure 64) provide the visual representations of data in the ACES Viewer. Renderers typically (but not always) require input from a Data Table or Transform to provide data to render. When a Renderer does require input, it will draw one visual representation of each Tuple in the input Table.



Figure 64 - Two Renderers included with the ACES Viewer

4.4.1.1.5 Expressions

An Expression is a simple function that accepts a Tuple as input, and produces an arbitrary output. Expressions provide a means of extracting relevant data from each Tuple in a Table that is providing input to a Transform or Renderer. A Transform or Renderer will typically require certain data from each Tuple in the input Table(s). These components will expose Expressions to users so that the user can specify what data in the Tuple provides the necessary information. For example, a Renderer that draws 3D Models at specified geographic locations will require latitude, longitude, and altitude values from each Tuple. This Renderer would provide Expressions for each required value (latitude, longitude, altitude). Each Expression is evaluated against every Tuple in the input Table to determine the location where each model should be rendered.

4.4.1.1.6 Controls

Controls (Figure 65) provide a graphical user interface to manipulate the parameters of other components that make up a visualization. For example, the ACES Viewer provides a Transform that filters data from an input Table based on a user specified time range. Only those Tuples whose time field values fall within the specified time range are included in the output of the Transform. The ACES Viewer also provides a Time Control. This control provides a user interface that allows users to dynamically manipulate the filter criteria (time range) of the Time Filter Transform.

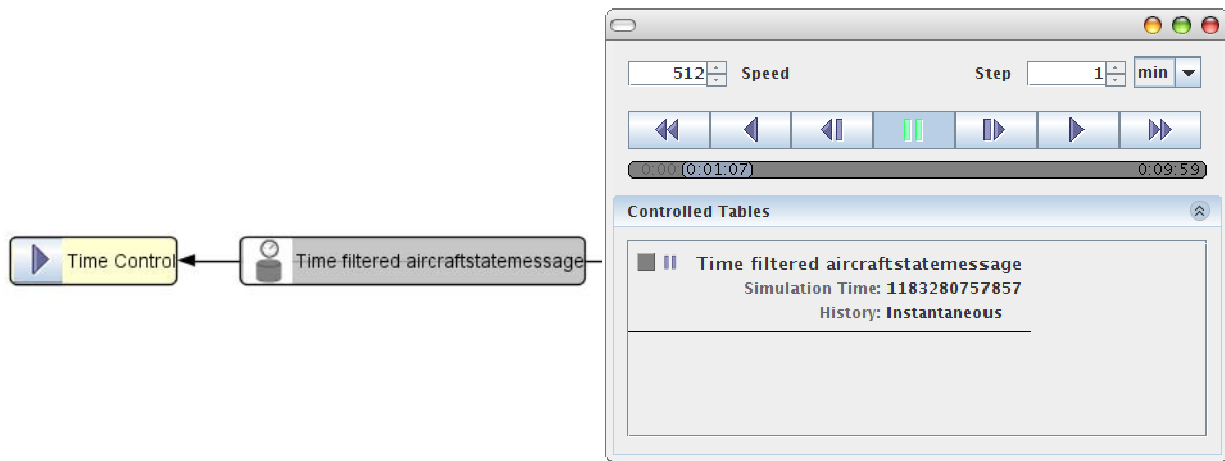


Figure 65 - The Time Control (right) and its corresponding representation in the Visualization Graph (left)

4.4.1.1.7 Visualization

A Visualization in the ACES Viewer is comprised of a collection of the components described previously, with the addition of Links that define relationships between the components. Figure 66 Left shows the Visualization Composition Graph, where graph nodes represent the various components of the visualization and the edges indicate data flow between the components. Figure 66 Right shows the resulting visualization.

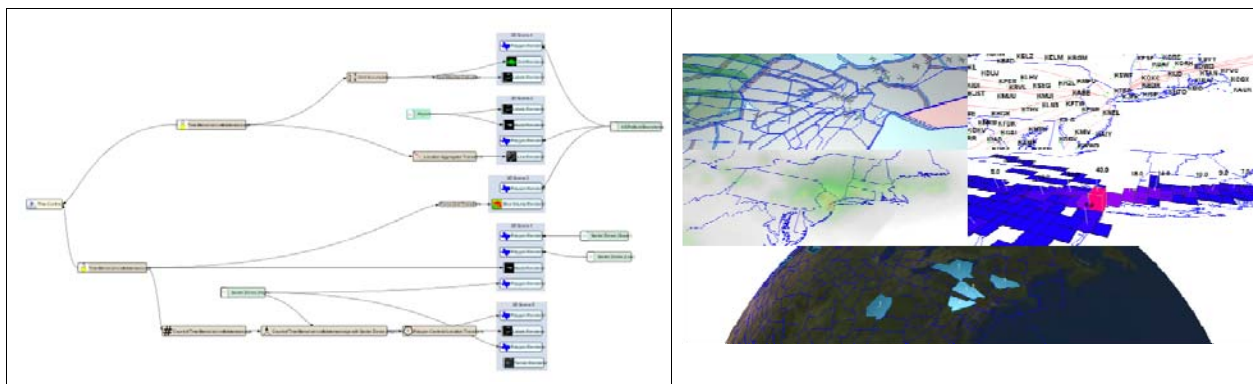


Figure 66 - A Visualization comprised of 5 displays and a variety of transforms, renderers, data tables and controls.

4.4.1.2 Component Model

The ACES Viewer uses the JavaBeans component model to integrate and control the various visualization components. JavaBeans is a Java component architecture commonly used when developing user interface components. JavaBeans imposes a limited number of requirements on the component implementation and provides the ability to utilize design-time meta data and Java's introspection mechanism to graphically manipulate component properties and

collaborators. This technique is frequently used by visual Graphical User Interface (GUI) builder tools. The process of assembling a GUI is very similar to one of the core ACES Viewer application functions which is assembling a visualization scene.

The JavaBeans specification standardizes various aspects of the components that adhere to it, which allowed us to develop a system to manage the visualization components in a generic manner. Some of the capabilities provided by the ACES Viewer's component management system include:

- Persistence. Any JavaBeans compliant component can be persisted to disk for long term storage using a custom XML based format.
- Automatic User Interface generation. Components may expose a variety of properties that control their behavior (or appearance in the case of visual components). The metadata provided with each component (part of the JavaBeans specification) allows us to automatically generate a user interface to manipulate these property values.
- Property value binding. The values of any JavaBeans properties on a visualization component can be defined externally and shared among multiple components. Changes in the property value cause *Property Change Events* to be issued and this allows us to synchronize the values of properties across multiple components, as the value changes over time.
- Component linking. Visualization components are connected together to form a data flow graph (data flows from Tables through Transforms, to the Renderers) using the property value binding mechanisms.

4.4.1.3 Concurrency

In order to take advantage of computer systems with multiple processors, which are becoming increasingly common, CACI designed and implemented a flexible concurrent I/O and processing system for the ACES Viewer. The system is comprised of a number of low level concurrency controls and utilities, and several task-specific thread pools that are automatically configured to match the capabilities of the host system.

- I/O pool: This is used for I/O intensive tasks. This type of task typically does not consume much CPU time, they mostly wait for other devices (hard disk, network, etc.) to complete a requested operation.
- CPU pool: This is used for CPU intensive tasks such as data processing, and some rendering calculations that can be performed prior to drawing.
- Control pool: This is used for control tasks. These are typically short-lived operations that must be executed in a timely manner (e.g. user interaction with the GUI).

Segregation of the I/O and CPU pools allow I/O and CPU bound tasks to execute concurrently, since there will typically not be too much resource contention between the task types. Each pool is allocated a small number of threads based on the number of available processors in the host system. Thus each pool can have at most one actively executing task per CPU. I/O threads are given a lower priority so that they are not scheduled for CPU time as often as the CPU threads. The control pool consists of a single thread with a high priority. Tasks given to the control pool for execution will be scheduled for execution quickly, even if other CPU intensive tasks are active in the CPU pool.

Any of the visualization components are free to use the appropriate thread pool as necessary. To simplify the implementation requirements of visualization components, we implemented a number of utilities to track task execution progress, including potential failures of individual tasks.

4.4.1.4 Extensibility

We have incorporated an extension mechanism to facilitate external development of visualization and datasource components designed to work with JView and the ACES Viewer application. Developers can write a visualization component using the public API of the ACES Viewer, and package it as a .jar file (separate from the ACES Viewer). This jar file, along with a small XML based metadata file can be placed in a subdirectory of the ACES Viewer binary application, where it will automatically be detected and integrated into the application. This allows the ACES Viewer's functionality to be extended without the need to recompile the entire application. In fact, developers do not even need access to the ACES Viewer source code to develop and integrate an extension.

4.4.1.5 Logging

The ACES Viewer includes integrated logging capabilities that record each task as it is executed in the application. This is primarily useful for debugging problems that may occur as we develop the application, and problems users encounter while using the application. The logged data is stored in an XML file that can be visualized with third party tools, or with the integrated log viewer provided with the ACES Viewer. The logs also include information regarding the interactions of users with the application, which can be used to evaluate the effectiveness of the user interface, and the application itself.

4.4.1.6 Image/Movie Capture

The ACES Viewer includes integrated image and movie capture functionality, allowing users to save visualization images for offline viewing. This provides the following benefits:

- Avoids system resource limitations. If the application is being run on a slower machine, or working with extremely large datasets then the visualization displays may have difficulty keeping up with the simulation playback at the desired speed. The movie capture allows frames to be captured independently of the speed that they will be played back.
- Allows visualizations to be presented without the application. There is a fairly large amount of data involved in playing back a simulation. With the ability to display visualizations of

multiple simulations, the size of the data can easily surpass 10 gigabytes. That is a lot of data to transfer when the interactivity of the application is not required. In this case, movies can be generated that only use a small fraction of the size of the dataset. These movies can be combined with and embedded in other media types as part of a presentation, for example.

- Allows application output to be presented by someone who is not necessarily familiar with the application interface. It is considerably easier to press a single play button on a media player application than it is to use the visualization application. This also eliminates startup and data loading time.
- Allows users to communicate problems. If, for example, a visual artifact appears in one of the visualization features under development, the user can submit a small video illustrating the problem to the application developers.

4.4.2 Data Access

4.4.2.1 Data Input

Access to data sources is provided through implementations of the Table interface of the ACES Viewer API. Developers that wish to access a particular type of data resource (for example, a file with a specific format) must provide a Table implementation that represents data records in the data source as Tuples. This can be accomplished in a number of ways:

- Using existing data access components. Certain common data formats are directly and generically supported by the ACES Viewer (presently JDBC data access and certain comma-separated values formatted files). In this case, the developer needs only to specify the type of data to access and where it is located (e.g. file path, database connection parameters, etc.).
- Pre-loading the data. The ACES Viewer contains a table implementation that allows users to add rows. The table will store all of the data in application memory. A developer could load all of the data from a resource into such a table. In this case, the developer is responsible for implementing the data access logic to be executed once the table is first created, to populate the table with Tuple records.
- Dynamic data access. Developers may choose to implement dynamic data access by providing a Table implementation that directly accesses its data rows from the source media (file/database/etc.). This allows a subset of the data to be loaded into application memory when requested, while the remainder is still stored in the source media. This type of access is required for extremely large data sources that will not fit in application memory.

Some of the data source types already supported by the ACES Viewer include:

- *Comma-Separated-Values* files. The ACES Viewer supports extremely fast reading and parsing of CSV data and conversion to Java types (from text) when a Schema specifying the data type of each CSV field is provided. Data can be read from disk as needed when

memory utilization is a concern or can pre-load all of the records into memory when performance is more important.

- Relational Database Data. We provide access to any data stored in a database system that has a Java Database Connectivity (JDBC) driver. Most open source and commercial databases include a JDBC driver. Data can be read from a single database table or from multiple tables by providing the SQL query to read it. Database types are automatically converted to Java types using metadata information provided by the JDBC driver. This also includes support for access to remote (networked) database systems.
- National Digital Forecast Data (NDFD) Gridded Binary (GRIB) Data. NDFD data forms the base of the National Oceanic and Atmospheric Administration (NOAA) National Weather Service Digital Services Program. NDFD data is freely available in gridded GRIB format from NOAA servers, which may be queried for a number of available fields, including wind direction, wind speed, temperature, precipitation amount, wind gust, etc. The data access components in the ACES Viewer support access to surface wind direction/speed and surface temperature data.

4.4.2.2 Data Output

The ACES Viewer supports the output of any Table instance in the application to a CSV file stored on disk, provided that the necessary components are present to convert the data records in the Table to a textual representation. This includes output of *processed* data derived from one or more Data Tables using a Transform component. Once the data has been saved to a CSV file, it can be loaded back into the application using the CSV Table reader, or used with other applications that support the CSV standard.

4.4.3 Data Processing

CACI has developed a number of data processing (Transform) components to derive information from Data Tables. The Transforms included in the ACES Viewer fall under the following categories: Relational Transforms, Filtering Transforms, Aggregating Transforms, and Special Purpose Transforms, which are described in the following sections.

4.4.3.1 Relational Transforms

4.4.3.1.1 Join Transform.

The Join Transform operates on two input Tables, performing an *inner join* using a user specified join criterion. The two input tables are differentiated as a left table and a right table. The values in the left and right tables will be included so long as there are matching rows (determined by the join criteria).

4.4.3.1.2 Concatenating Transform.

The Concatenating Transform concatenates the Tuples from multiple input Tables together producing output where the number of Tuples is the sum of the number of Tuples in each of the input Tables. The input Tables must have identical Schemata.

4.4.3.2 Filtering Transforms

4.4.3.2.1 Filter Transform

The generic Filter Transform uses a single boolean expression to determine whether Tuples in a single input table should be included in the Transform output or not.

4.4.3.2.2 Time Filter Transform

The Time Filter Transform is similar to the generic Filter Transform; however the filter criterion is based on a numeric time value and a history duration, which together define a range of times. Any Tuples in the input that have a time that falls within this range are included in the output. This Transform can be used in conjunction with a Time Control for animated playback of time-keyed data.

CACI has also implemented a specialization of the Time Filter Transform designed for use with database Tables. This specialization uses SQL queries to evaluate the filter criteria for improved efficiency.

4.4.3.3 Aggregating Transforms

4.4.3.3.1 Aggregator Transform

The Aggregator Transform collects values from multiple input Tuples into a single output Tuple (there can be multiple output Tuples). The output Tuple represents a simple aggregation of the input Tuples by collecting all of the values in the input Tuples into arrays of values. Grouping is specified by a user specified Expression; each input Tuple where the Expression evaluates to the same value is grouped into a single output Tuple.

4.4.3.3.2 Grid Accumulator Transform

The Grid Accumulator Transform divides a 2D (latitude/longitude) region into a regular grid and calculates the number of locations in the input data that fall within each grid cell. The location and span of the region is automatically calculated to include all of the locations in the input data. For performance and memory efficiency concerns, this Transform breaks the region down into multiple sub-regions, each with a corresponding numeric count array and bounding box. The number of sub-regions can be controlled via properties of the Transform, as can the resolution of the grid. This transform is designed for use with the Grid Renderer.

4.4.3.3.3 Force Grid Transform

The Force Grid Transform produces a 3D numeric array representing the intensities of a field sampled at discrete intervals (defined by the grid resolution). Input data consists of a set of locations and for each grid cell, the intensity value is the sum of $QValue/distance^2$ for each location in the input (distance is the distance from each location to the center of the grid cell). This Transform is designed for use with the Slice Volume Renderer.

4.4.3.3.4 Counting Transform

The Counting Transform produces a count of input Tuples where the user specified key Expression evaluates to the same value.

4.4.3.4 Special Purpose Transforms

4.4.3.4.1 Grid Maxima Calculator

The Grid Maxima Transform computes the values and locations of local maxima of 2D grids provided as input. This is designed for use with the Grid Accumulator Transform.

4.4.3.4.2 Polygon Centroid Transform

The Polygon Centroid Location Transform computes the centroid (in terms of latitude, longitude, and altitude) of polygons in the input Table.

4.4.3.5 Table Events

Most Transforms expose parameters that may affect the data processing thereby changing the data represented by the Transform. In order to respond to these changes (for example updating the visual output of a renderer to reflect new data), the Table API exposes methods that allow registration of a listener object that will receive notification when the content of a Table (or Transform) changes. Using this facility, components that accept input data from a Table/Transform typically monitor the input source for changes so that they might take an appropriate action.

4.4.3.6 Expressions

In order to increase the flexibility of the renderer and transform components in the ACES Viewer with regard to input data, Expressions provide user tunable per-Tuple processing capabilities. Expression implementations are given an input Tuple to evaluate, and return a single object whose type is determined by the expression implementation. As an example, the location model renderer (which draws a user specified geometry at multiple locations) has an associated *location expression*. The location expression is evaluated on each row of an input table to extract a location (in terms of latitude, longitude, and altitude) at which the geometry will be rendered. This decouples the location model renderer from the schema of the input data; the expression is responsible for interpreting the input rows to extract the necessary information. The location model renderer could then support alternate input data formats (for example an input table might only have latitude and longitude columns and all altitudes are assumed to be 0) simply by replacing the location expression.

Expressions are used by both Transform and Renderer components. Other example uses of expressions include the filter predicate for the Filter Transform (the filter predicate returns a boolean value for each input row indicating whether it passes the filter), a color expression for the polygon renderer (each polygon can be colored individually based on the color value indicated by the expression).

The ACES Viewer provides 3 basic types of Expression:

- **Literal:** The Expression evaluates to the same value regardless of the Tuple used as input.
- **Data Field:** The Expression evaluates to the value of a single column (with a compatible data type) in the input Tuple.
- **Groovy Expression:** This is a generic Expression that allows users to write the function implementation in the Groovy scripting language. All columns of the input tuple are passed as arguments to the user defined function and the implementation (written by the user) determines the output.

4.4.3.7 Controls

Controls provide an interface to manipulate parameters of visualization components (Transforms in particular). A Control's user interface is typically graphical. Presently, the ACES Viewer has a single control which is the Time Query Control (Figure 67). This implementation works in conjunction with a Time Filter Transform (or other components that implement the TimeTable interface).

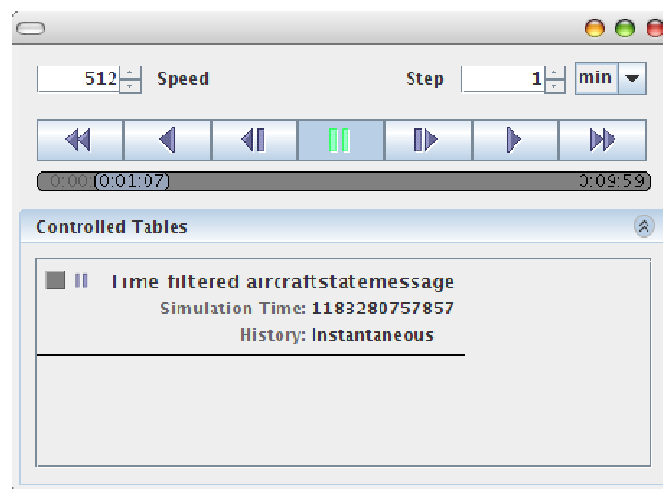


Figure 67 - Time Query Control

This Time Control provides VCR-like playback controls as its user interface for controlling the time and extent parameters of the Time Filter Table. Each Time Control can control any number of Time Filter Transforms. This allows the time and extent to be synchronized among multiple Time Filter Transforms. The Time Filter Transform uses the time and extent values provided by the Time Control to filter input data based on a time key contained in each data row. By changing the parameters, the Time Filter Transform's content is modified to contain only data rows that pass the filter criteria. This control uses the ACES Viewer's concurrency utilities to track the cascade of updates that occur when the time is advanced, which ensures that all of the

components in a visualization graph have had a chance to respond to changes in the Time Filter Transform's content before the time is advanced again.

4.4.4 Visual Components

4.4.4.1 Model Renderer

Figure 68 shows one of the first JView based visualization components developed for the ACES Viewer, showing the location and heading of simulated aircraft. This component is designed to display any data represented by a location (latitude, longitude, altitude) and optionally an orientation (yaw, pitch, roll). Figure 69 and Figure 70 show other events drawn with the Model Renderer.

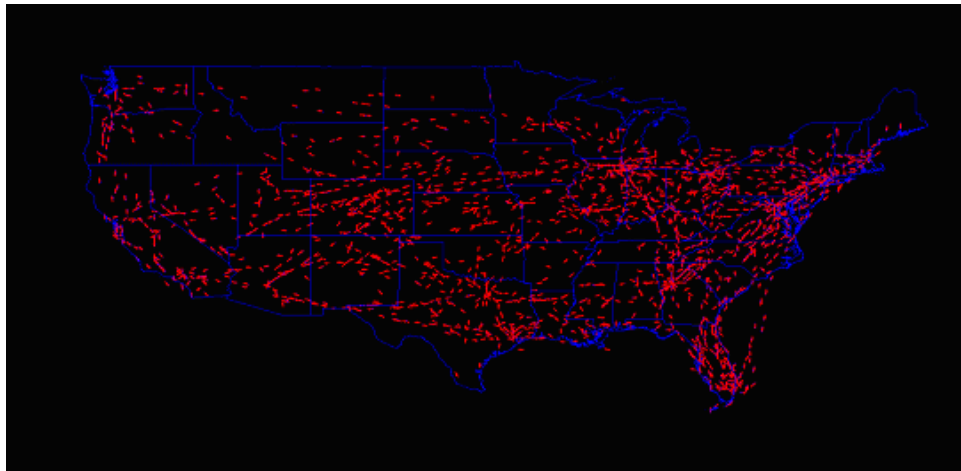


Figure 68 – Individual Aircraft Display

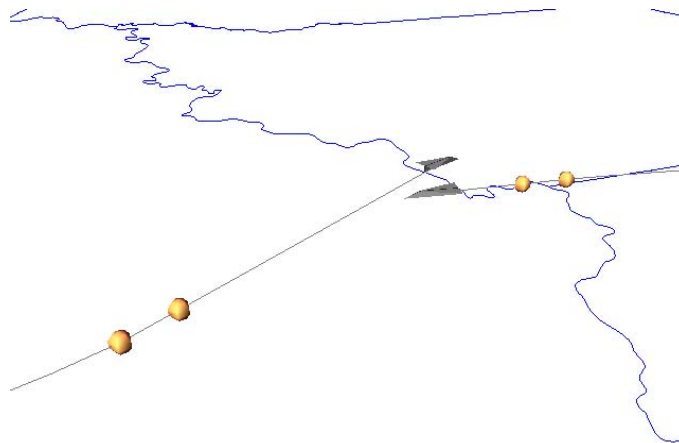


Figure 69 - Maneuver Locations drawn with the Model Renderer

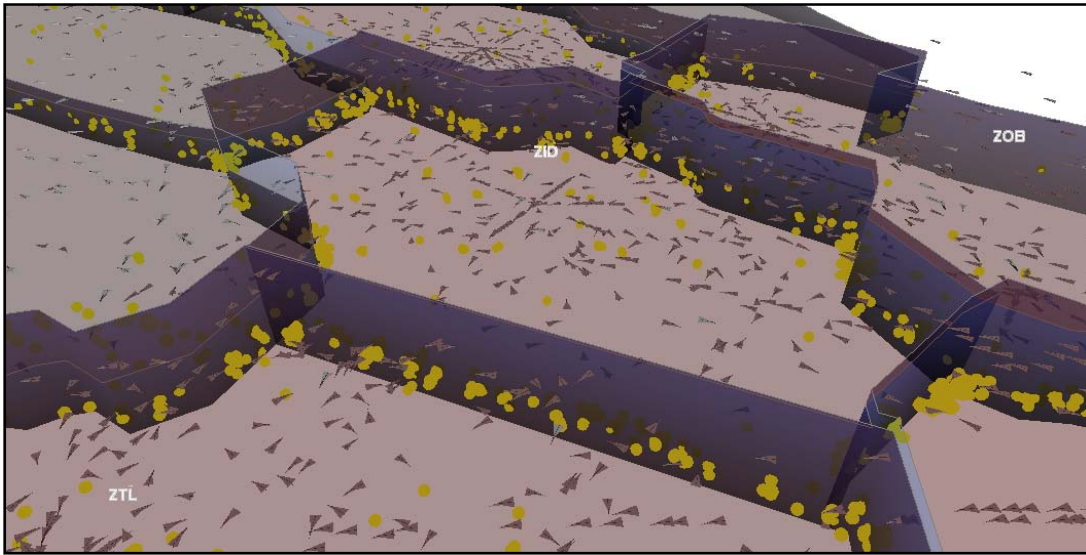


Figure 70 - Center Boundary Crossing Feature (Yellow Spheres)

The initial work was quickly enhanced to include the following functionality:

- User specified geometry for each location (Figure 71)
- Efficient view-dependent rendering. The Model Renderer is designed to render a single geometry in multiple locations using a technique known as instancing. This technique amortizes the cost of preparing the graphics hardware for rendering the geometry over each position that it will be drawn in. We also use view-frustum culling to limit the number of models that will be rendered to only include those that will be visible given the camera position and orientation.
- Auto-scaling of geometry. The Model Renderer can be configured to render the model geometry at a fixed size in screen space, by inverting the scaling that is applied by the perspective projection when the models are a significant distance from the camera. Figure 72 illustrates this capability, where similarly colored spheres occupy the same amount of screen space, regardless of their depth in the scene.
- Geometry appearance modification. The Model Renderer can modify certain appearance properties of the geometry, such as its color, whether lighting is enabled, and the polygon rendering modes (for wireframe rendering). This is accomplished by traversing the geometry tree structure and inserting modifier nodes between specific pieces of the geometry. Figure 73 shows the paper-airplane geometry with modified material properties (it is normally grey).
- Projection support. The Model Renderer can use any JView Projection implementation to convert locations and orientations relative to a latitude, longitude and altitude to a

Cartesian location. JView currently provides Mercator, WGS84 and Lambert conformal Projections.

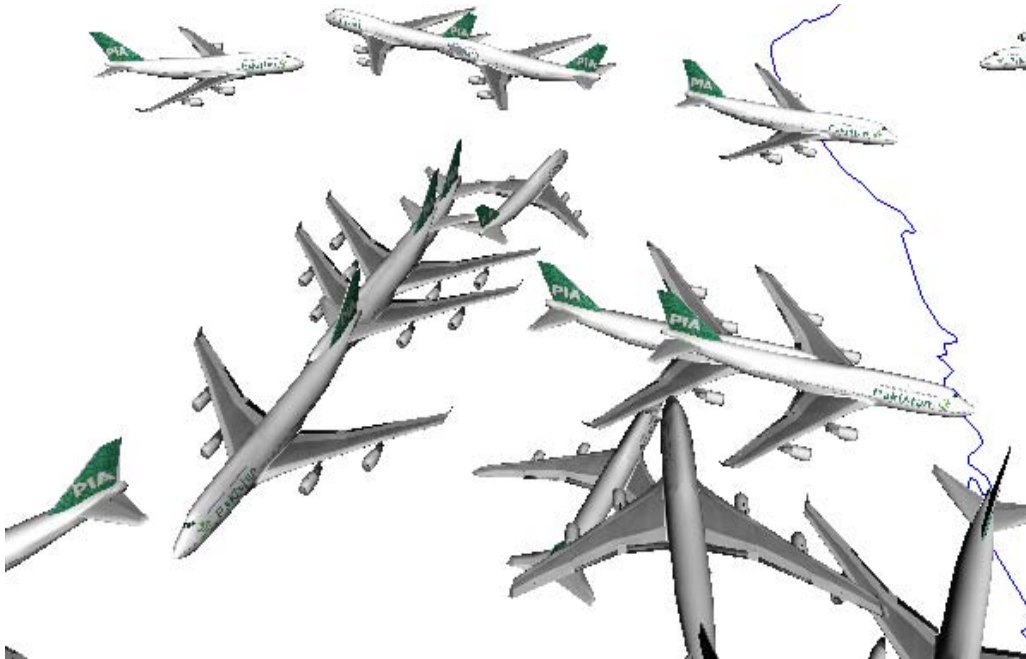


Figure 71 - Aircraft Models Rendered by the Model Renderer

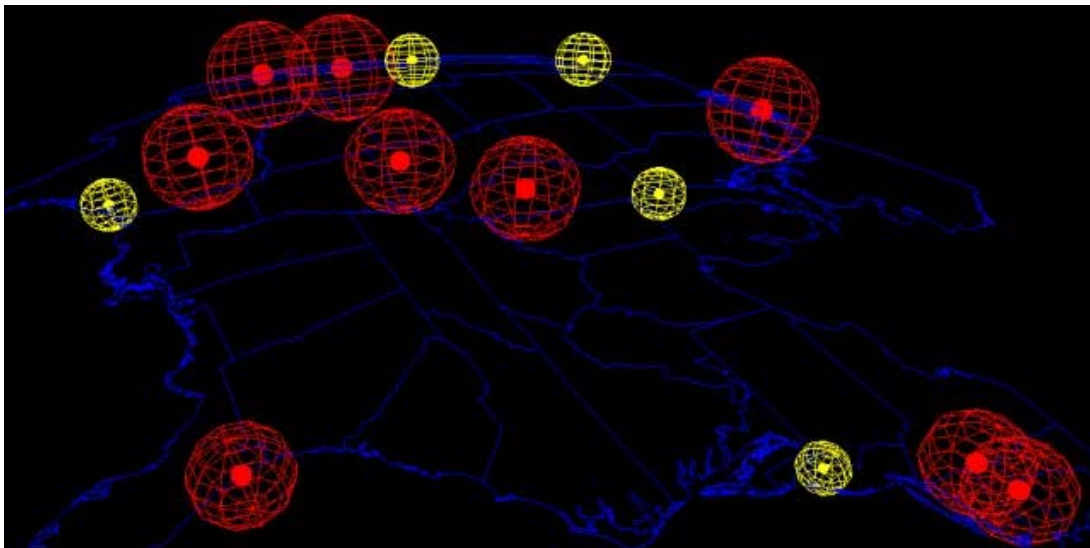


Figure 72 - Location Model Feature Autosize

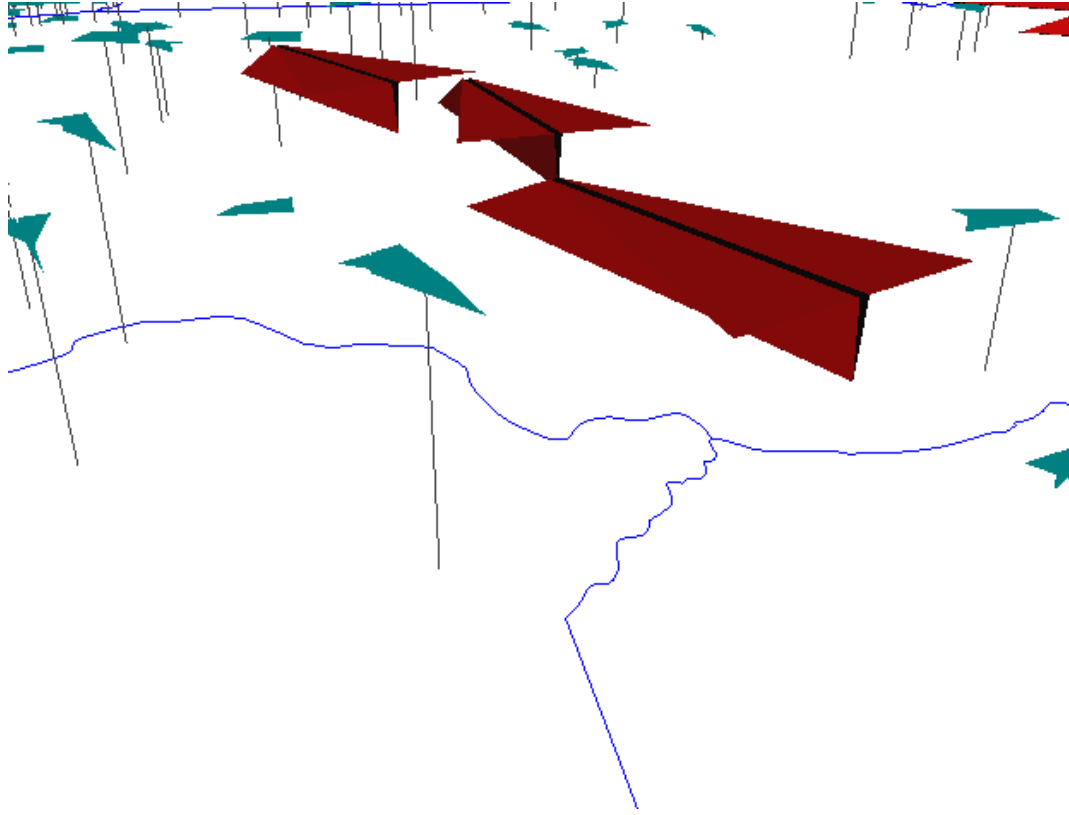


Figure 73 - Aircraft Model Feature with Different Rendering Properties

4.4.4.2 Line Renderer

The Line Renderer, as the name implies, draws lines between sequences of points. This can be used to draw trajectories, flight plans, network links, and other data types consisting of sequences of two or more locations. The Line Renderer, as shown in Figure 74, provides the following capabilities:

- User specified colors
- Support for dashed lines
- Support for animation with dashed lines
- Leading and/or Trailing alpha gradients
- Line Smoothing

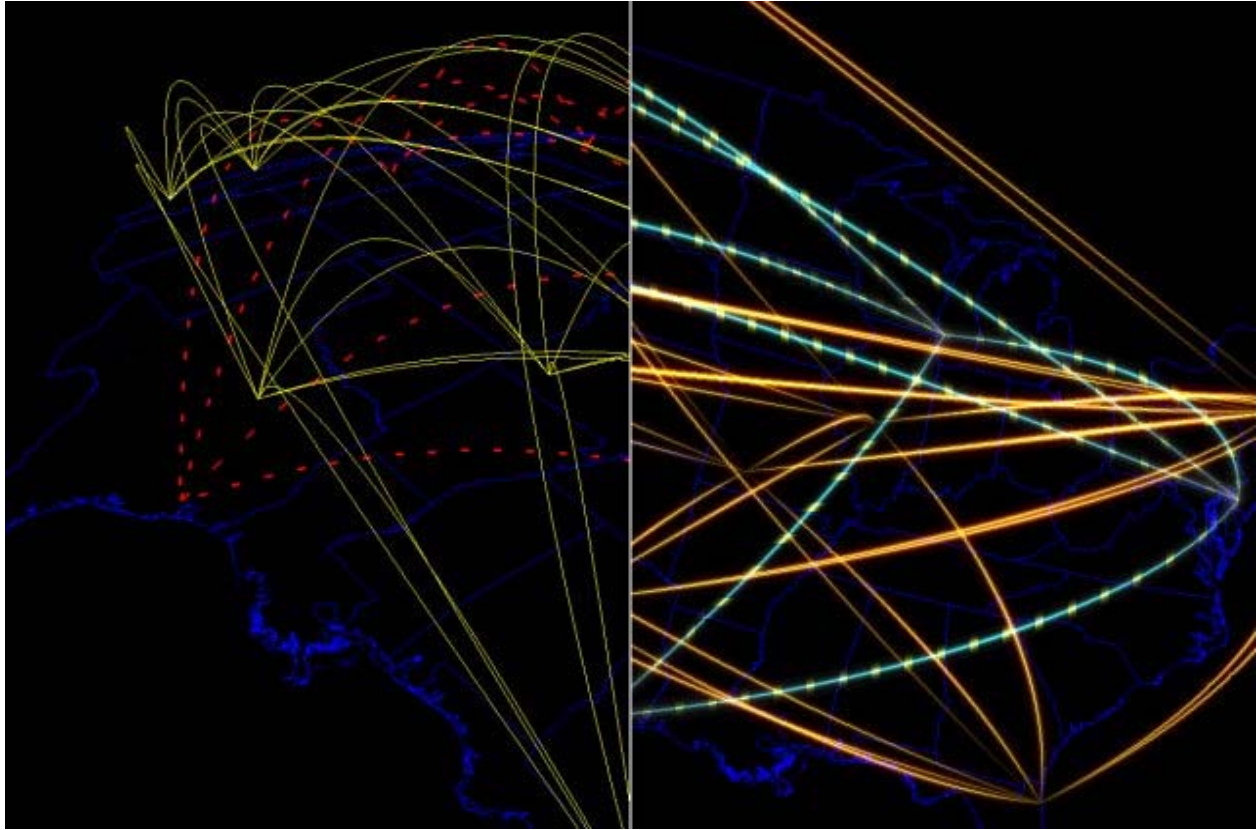


Figure 74 - Line Renderer

Figure 75 shows the Line Renderer used to draw conflict trajectories. Figure 76 shows aircraft trajectories and Figure 77 shows a network visualization based on the number of aircraft moving between airport pairs.

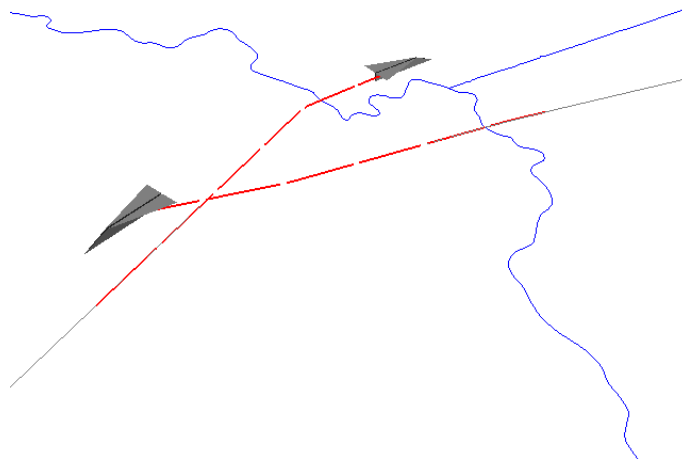


Figure 75 - Conflict Detection Feature (red lines), along with aircraft models, and trajectories

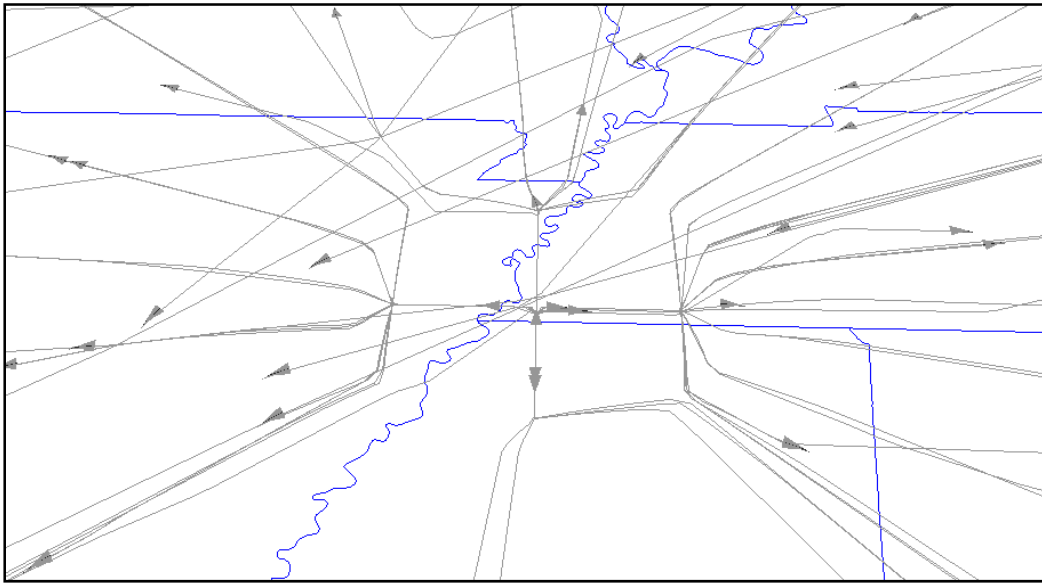


Figure 76 - Aircraft Trajectory Visualization

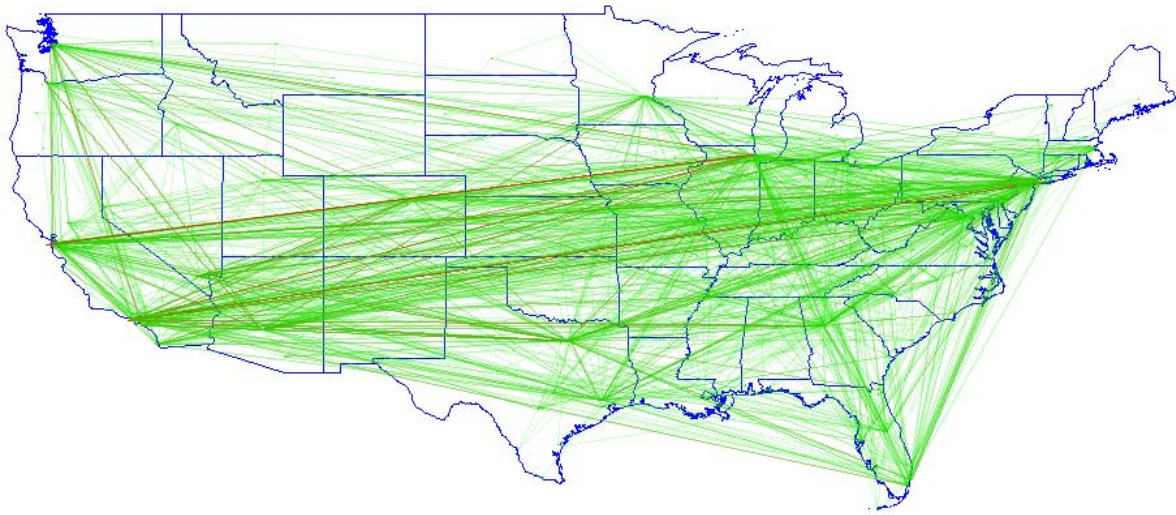


Figure 77 - Airport Network Feature

4.4.4.3 Slice Volume Renderer

CACI developed a component to render volumetric data using OpenGL's 3D texture support. The Slice Volume Renderer uses a transfer function to map intensity values specified by the user to a color and opacity value. The resulting colors are compiled into a 3D texture which is transferred to the graphics hardware. The renderer then draws a number of view-aligned parallel *slices* through the region occupied by the volume, using a variation of the Marching Cubes algorithm for iso-surface extraction (Figure 78).

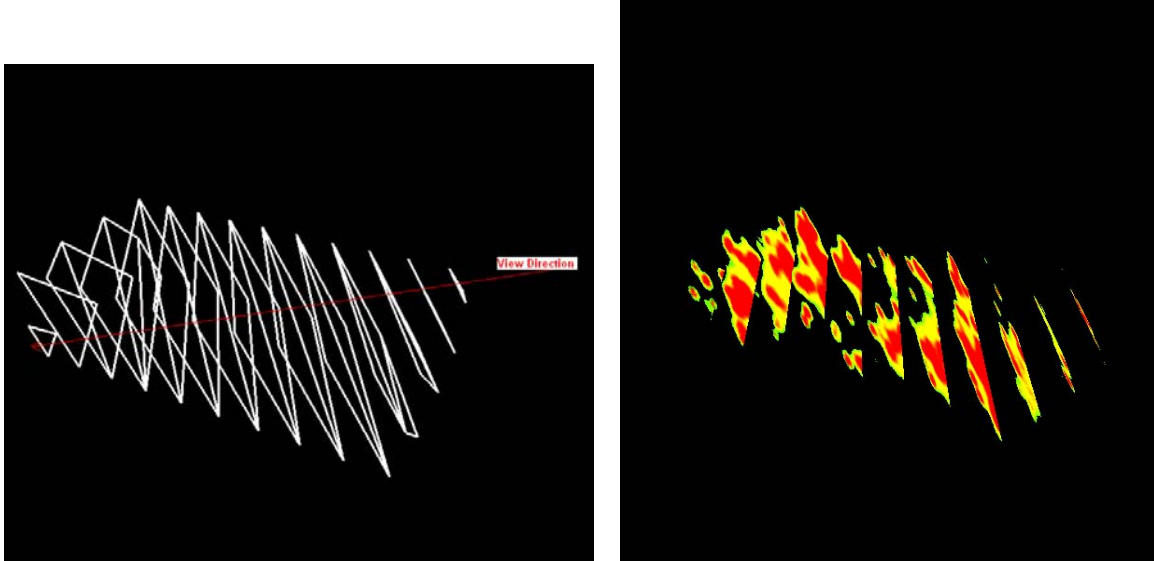


Figure 78 - View Aligned Slices, side view: Wire frame (left), Textured (right)

The slices are then texture-mapped so that the colors in the 3D texture are rendered at their appropriate locations in space (Figure 79 and Figure 80).

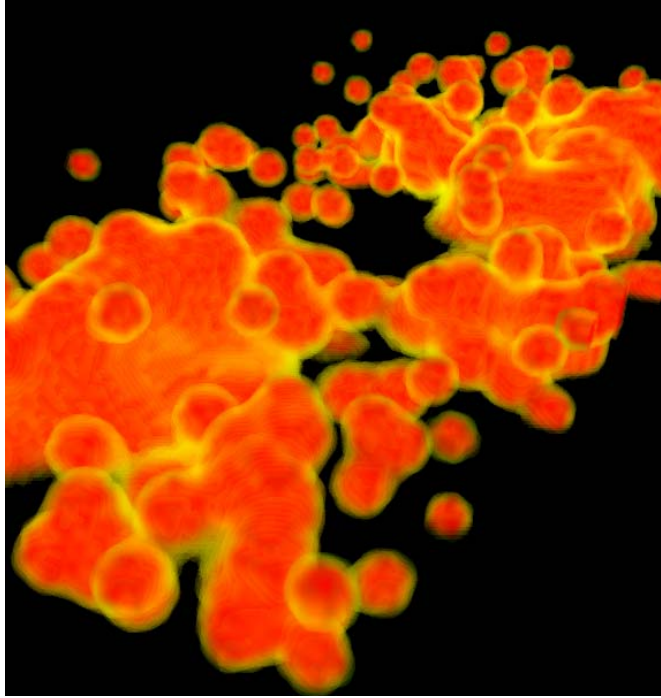


Figure 79 - The final blended image from the camera perspective

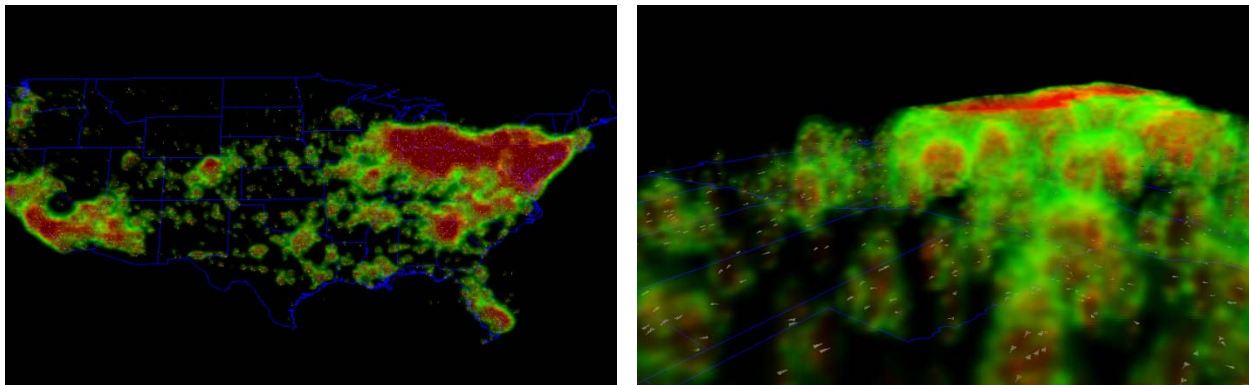


Figure 80 - The VAMS Aircraft Density Field Visualization

The Slice Volume Renderer provides the following functionality:

- Volumetric rendering of data stored in axis-aligned spatial grids.
- User defined transfer function to map intensity values to color and opacity values.
- User defined sampling rate, specified in samples per grid cell (this controls the distance between slice planes, and thus the resolution of the visual volume representation).

The Slice Volume Renderer provides basic volume rendering capabilities and allows for interactive visualization of volume data; however it does have some limitations:

- The volume data must be structured as a regular 3D grid of intensity values. Irregular, multi-resolution, and non-gridded (continuous/functional volume) data is not supported.
- The spatial boundaries of the grid must be axis-aligned. This also means that the volume must be rectilinear.
- 3D texture dimensions are limited by the graphics hardware. This restricts the dimensions of intensity grids that can be rendered.

4.4.4.4 Splat Volume Renderer

To address some of the limitations of the Slice Volume Renderer, we developed another volume renderer that uses *splats* to represent the volume data. Instead of using slices and 3D textures, this renderer draws a large number of point-splats, representing the color contribution of each volume region (voxel) to the total volume representation. Each splat is drawn using a *point sprite* with a Gaussian texture map in the alpha channel of the texture color, as shown in Figure 81.

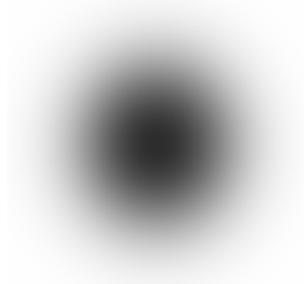


Figure 81 - A Point Splat

The volume region is traversed using an octree traversal algorithm. When the traversal has gone deep enough into the octree so that the volume represented by the octree nodes take up a small enough area when projected to the screen, a splat is drawn centered in the node. Once the entire volume has been traversed in this manner, the splats are blended to produce the final view of the volume (Figure 82).

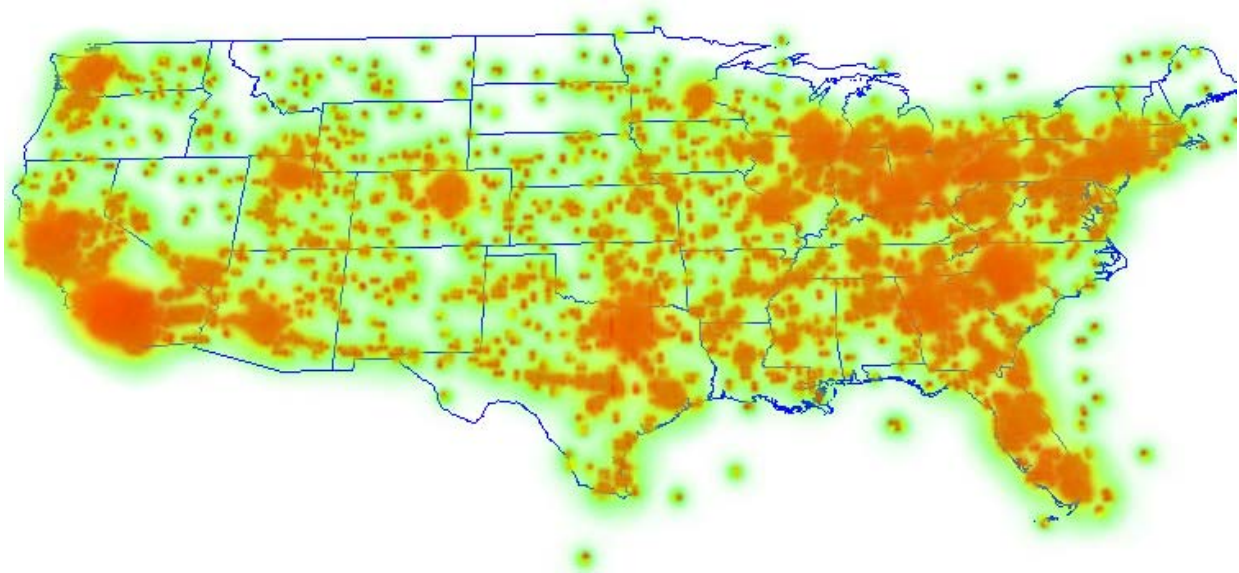


Figure 82 - Splat Volume Renderer

The Splat Volume Renderer avoids all of the limitations of the Slice Volume Renderer and also allows lighting effects to be incorporated into the volume visualization.

4.4.4.5 Histogram Bar Renderer

We developed a Renderer to display individual histogram bars in a 3D scene. This renderer differs significantly from the Histogram Grid Renderer described earlier. The Histogram Bar Renderer (Figure 83) is designed to display a relatively small number of histograms with significantly more control over their appearance compared to the Histogram Grid Renderer.

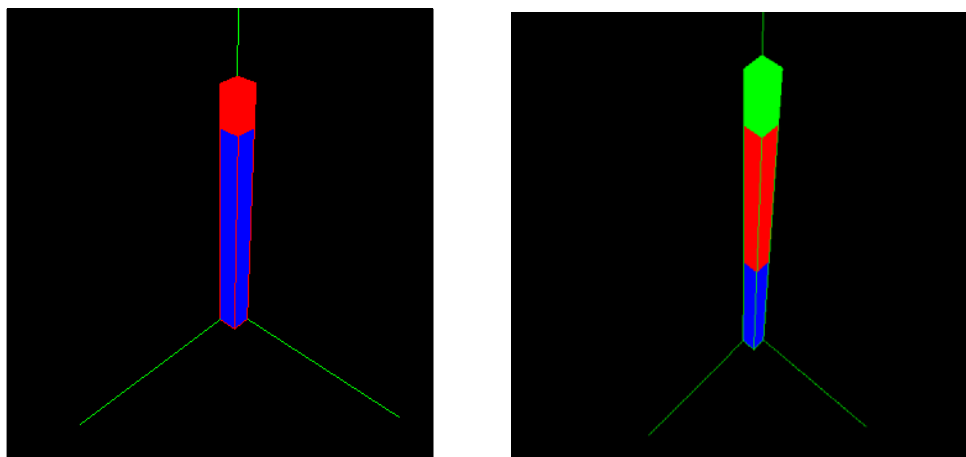


Figure 83 - Histogram Bar Renderer

Some of the capabilities of this renderer include:

- Multiple colors at user definable heights, which can be stacked to show multiple (related) values in a single histogram.
- View alignment options to ensure the order of multiple histogram bars (from left to right) remains constant from any viewing direction, while still providing a 3D component.

Several different configurations of this the Histogram Bar Renderer are shown in Figure 84.

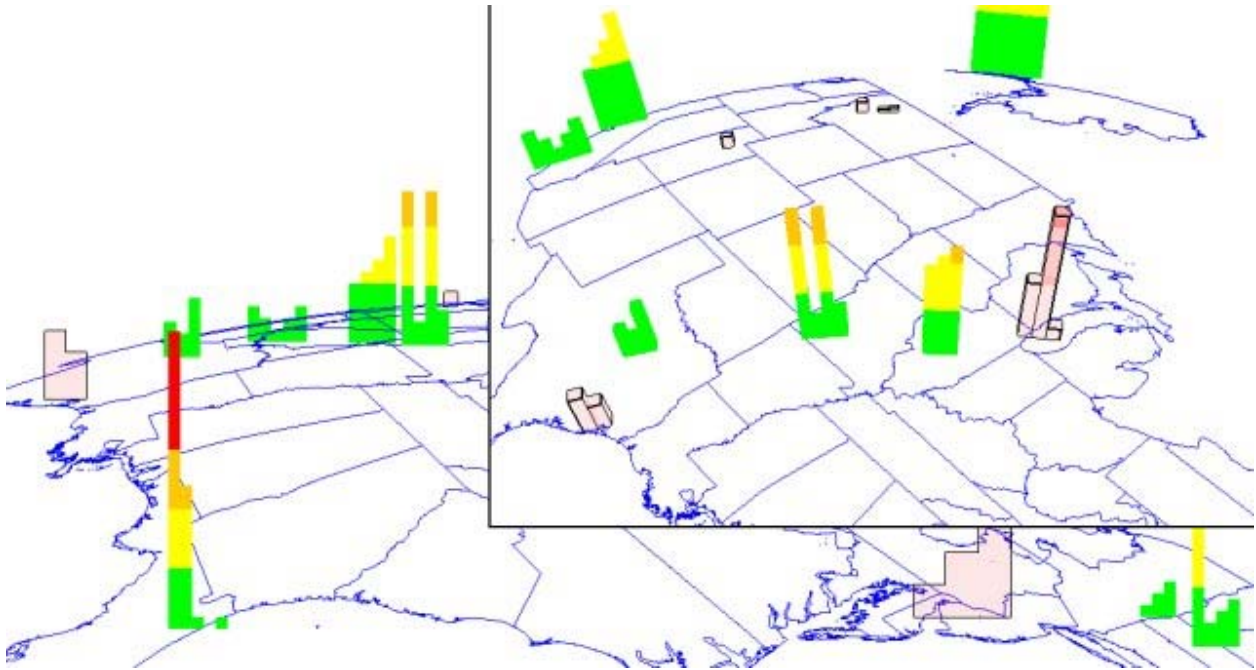


Figure 84 - Histogram Bar Renderer Configurations

4.4.4.6 Flow Field Renderer

The Flow Field Renderer displays heading and intensity values arranged in a 2-dimensional spatial grid. Each heading/intensity value pair is represented by an arrow, where the heading is encoded by the direction of the arrow, and the intensity is encoded in the color, size, and animation rate of the arrow (Figure 85).

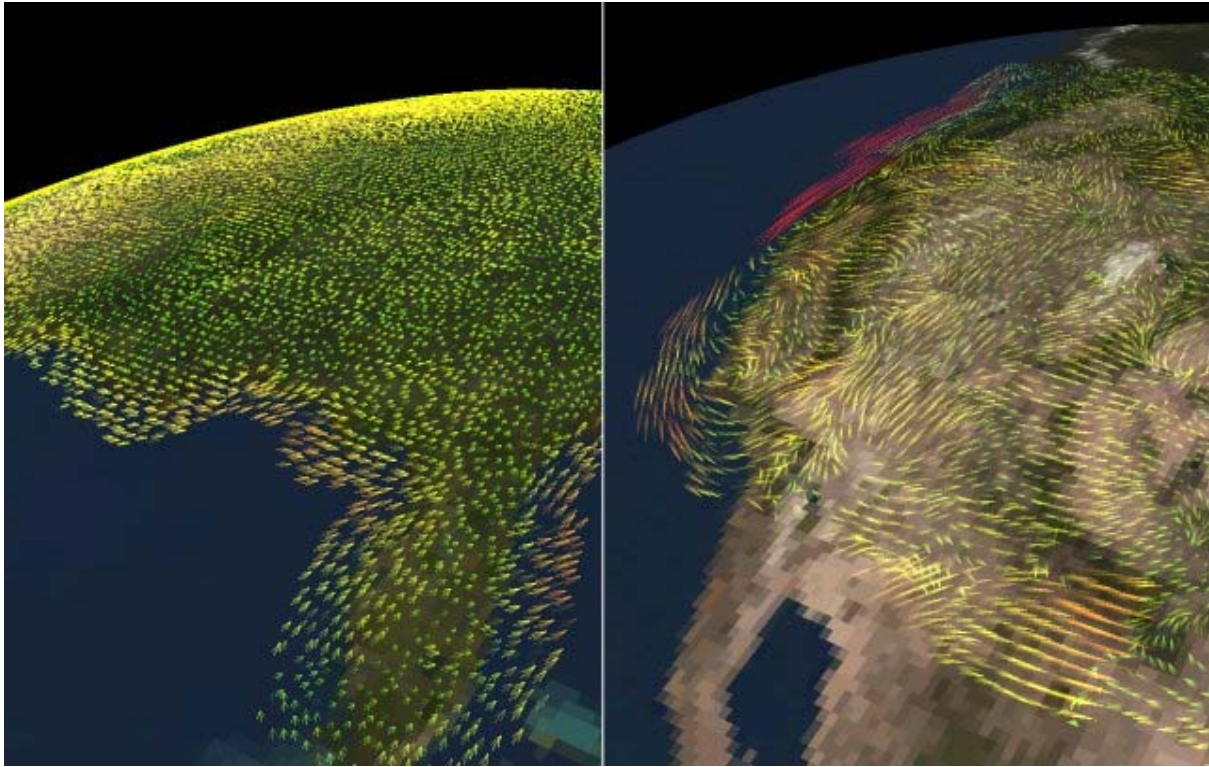


Figure 85 - Flow Field Renderer

The Flow Field Renderer provides the following capabilities:

- Optional Animation, the arrows move in the direction specified by their heading, at a rate controlled by the intensity value.
- User specified transfer function to map intensity values to colors.
- Support for arbitrary projections (Figure 86).

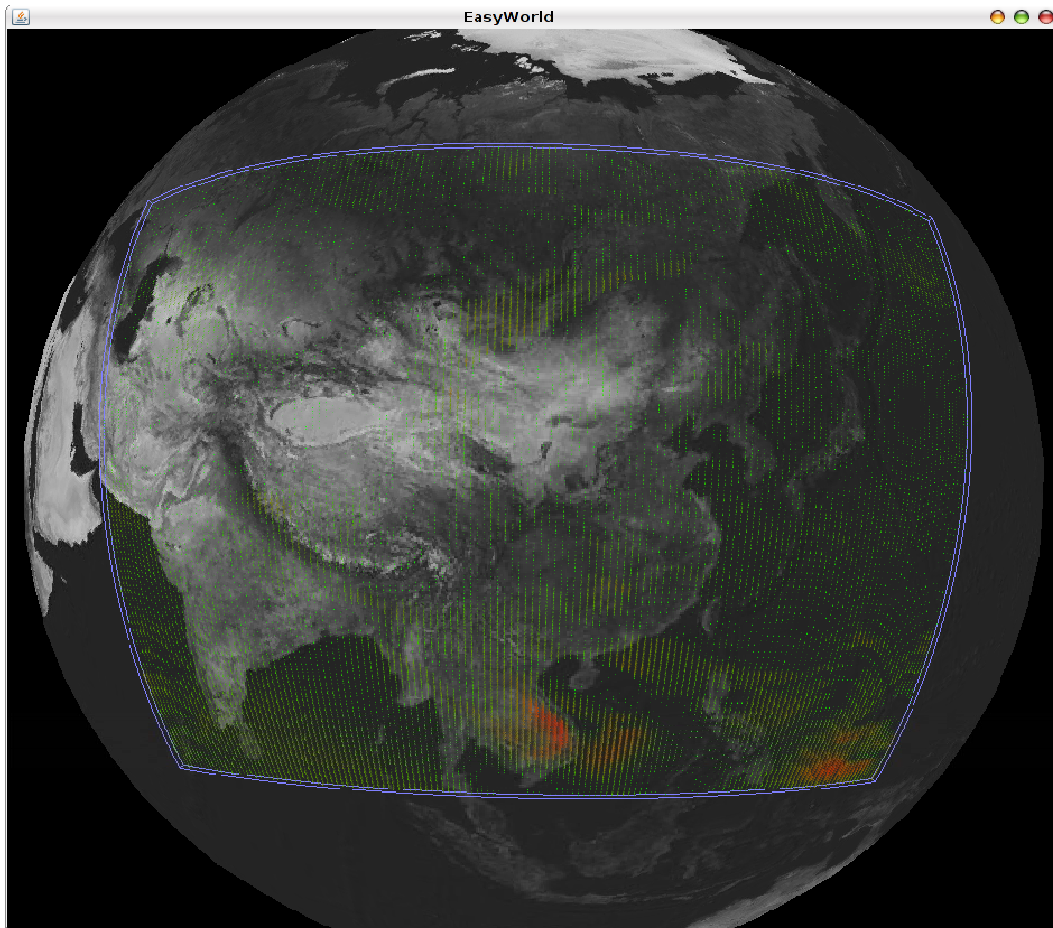


Figure 86 - The Flow Field Renderer drawing wind direction and intensity over Asia

4.4.4.7 Frustum Renderer

The Frustum Renderer is primarily useful as a debugging tool and shows a visual representation of the view frustum from a given camera location (Figure 87)

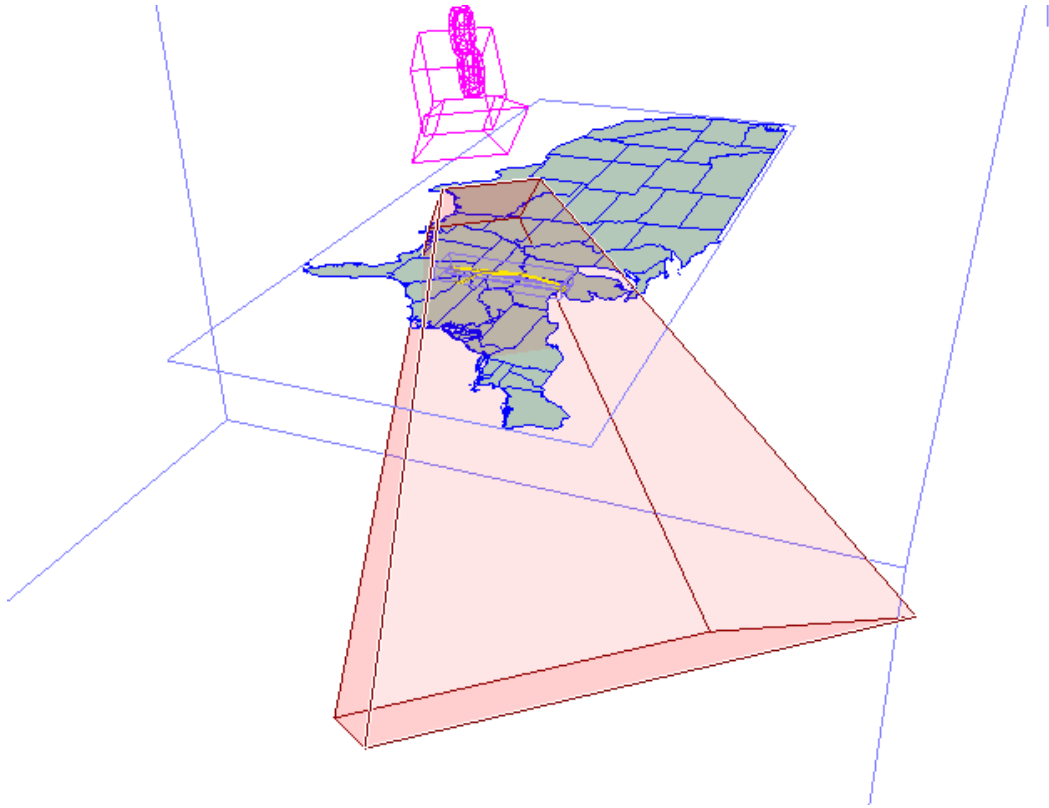


Figure 87 - A 3D scene displaying the bounding boxes of several visualization features. Also includes a view of the camera frustum

4.4.4.8 HUD Table Renderer

The HUD Text Renderer that CACI developed for JView allows users to display static text on the image plane of a 3D visualization. We have also developed an ACES Viewer specific extension of this component, the HUD Table Renderer. The HUD Table Renderer allows users to display the content of data tables or transforms on the image plane. Each row in a Table linked with the HUD Table Renderer will cause a single line of text to be rendered. Figure 88 shows the HUD Table Renderer displaying the number of aircraft in each sector zone along the right hand side of the display.

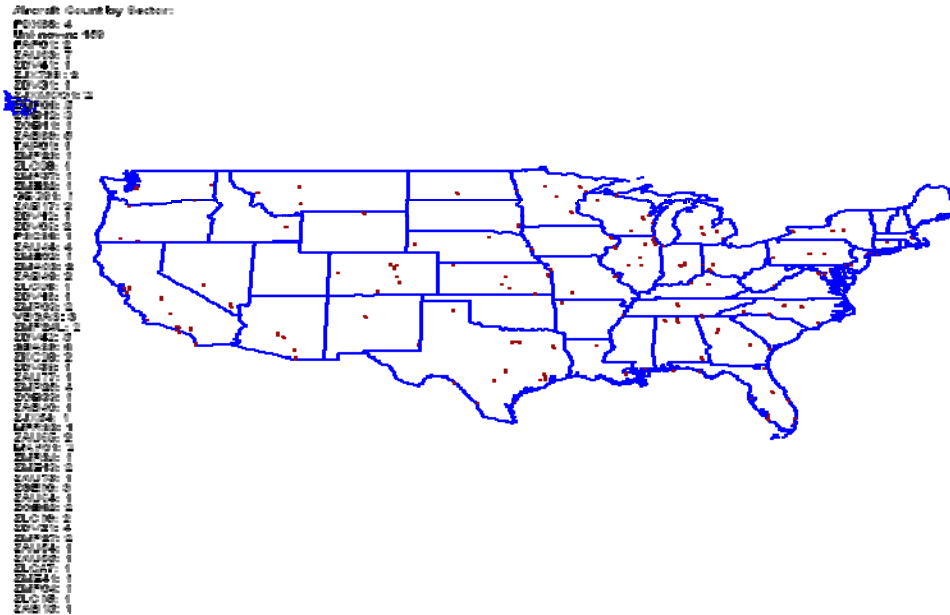


Figure 88 - Table Renderer displaying the number of aircraft in each sector zone

4.4.4.9 Display Containers

The ACES Viewer contains 2 Display Container implementations that provide a display area for Renderers to draw in.

4.4.4.9.1 3D Scene

The 3D Scene Display Container uses JView to provide an OpenGL based drawing area. This Display Container provides a number of services above and beyond what is included in JView's Graph3D drawing area, such as:

- Automatic management of Renderers that include a spatial bounding box defining the 3D volume in which they draw. This information is used to set up the initial camera position, and to manage clipping plane distances.
- Camera Navigation Support. The 3D Scene Container uses the Camera Navigation System that we developed for JView to allow users to move the viewing position in the scene.
- Projection Support. Each 3D Scene can have a single Projection associated with it, which is passed on to all Renderers that support projections. This ensures that all Renderers in a scene use the same map projection (if any).
- Stereoscopic rendering support. When enabled, 3D scenes can be rendered in anaglyph stereo or can produce two separate full-color images for the left and right eyes (for use with hardware supported stereo).

4.4.4.9.2 Tabular Scene

We developed a prototype Display Container to display the content of Tables in tabular form (Figure 89).

ABE	2.0
ABQ	6.0
ACY	1.0
ADS	1.0
AGS	3.0
ALB	6.0
APA	2.0
ATL	77.0
ATW	1.0
AUS	10.0
BDL	9.0
BED	2.0
BFI	2.0
BFL	1.0
BCM	1.0
BHM	11.0
BIL	1.0
BIS	4.0
BJC	1.0
BMI	1.0
BNA	15.0
BOI	6.0
BOS	20.0
BTR	2.0
BUF	5.0
BUR	1.0
BWI	25.0
CAE	4.0
CAK	2.0
CGF	2.0
CHA	4.0
CHO	1.0
CHS	3.0
CID	5.0
CLE	14.0
CLL	1.0
CLT	38.0
CMH	12.0
COS	7.0

Figure 89 - A prototype tabular scene with a visualization feature that displays the number of aircraft currently enroute to each airport

4.4.5 User Interface

4.4.5.1 Visualization Composition

The Visualization Composition Interface (Figure 90) is the primary user interface component for building visualizations in the ACES Viewer.

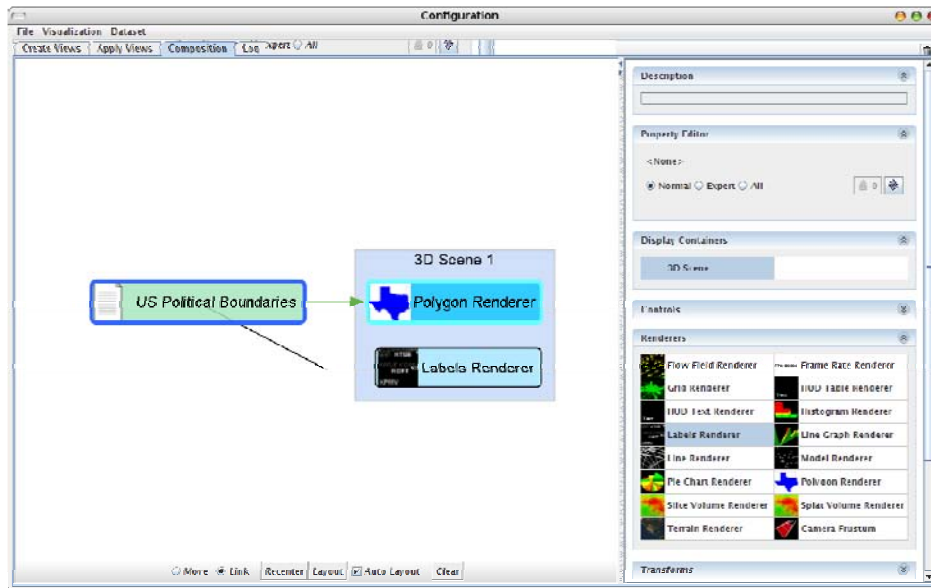


Figure 90 - The Visualization Composition Interface

The left side of the Visualization Composition Window consists of the Composition Graph. This shows an interactive live representation of the components that make up a visualization. Users can add or remove components, link components together, or select components to manipulate their properties. Users can also copy and paste components that are already present in the graph. The right side of the Visualization Composition Window contains task panes with all of the available visualization components (Controls, Renderers, Display Containers, Transforms and Data Tables). Users can drag components from the task panes into the graph area to add new components to the visualization.

After defining a visualization, it can be saved to disk so that it can be loaded by the ACES Viewer at a later time (Figure 91). The visualization is saved in an XML format that contains descriptions of all of the components present in the visualization, their property values, and representations of links between the components. Users can load the visualization and optionally apply it to different Data Tables than those that were present when the visualization was saved.

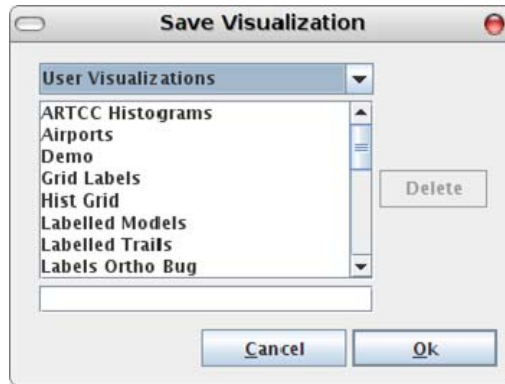


Figure 91 - The Save Visualization Dialog

4.4.5.2 Property Editor

When a user selects a node in the Composition Graph, the Property Editor task pane shows all of the user-controllable properties of the selected component (Figure 92).

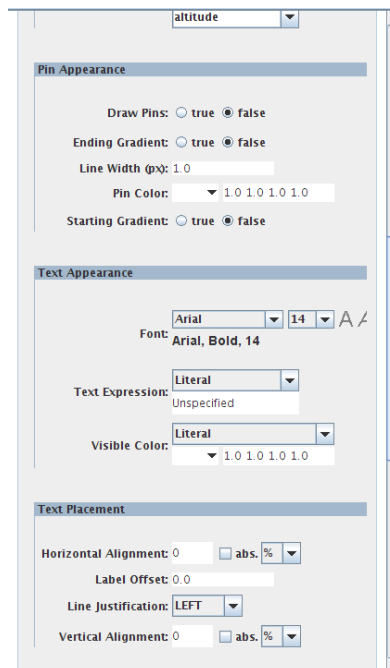


Figure 92 - The Property Editor

The Property Editor user interface is automatically generated by the ACES Viewer for each component using metadata provided by the BeanInfo classes associated with the visualization components (part of the JavaBeans specification). The metadata includes information on which properties should be available for editing by the user, human-readable property names and

descriptions, and optionally information on what type of user interface component is best suited for editing the property value. If no information is provided describing the user interface for a particular property, one is automatically chose by the ACES Viewer based on the data type of the property. Changes made in the property editor are immediately applied to the component being edited.

4.4.5.3 Dataset Definition

Users can load and manage Data Tables using the Datasets task pane and the Dataset Definition wizards. The Datasets task pane shows a listing of the loaded Data Tables and provides controls to load, unload, and define datasets (Figure 93).

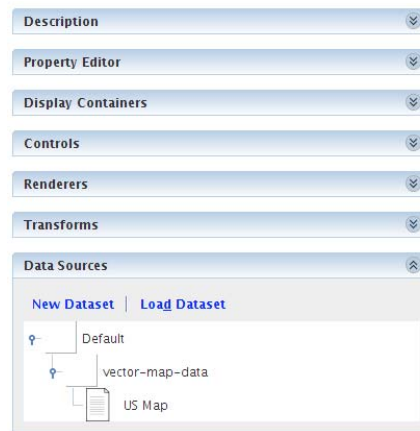


Figure 93 - The Datasets task pane

To load a data source into the application, users first activate the *New Dataset* control in the Datasets task pane. This action launches a wizard that allows users to specify the necessary information to load the data.

The first page of the wizard asks the user what type of data they would like to access. The wizard presents a list of supported data types with a brief description of each (Figure 94). The supported types that are available depend on the extensions that are installed in the ACES Viewer.

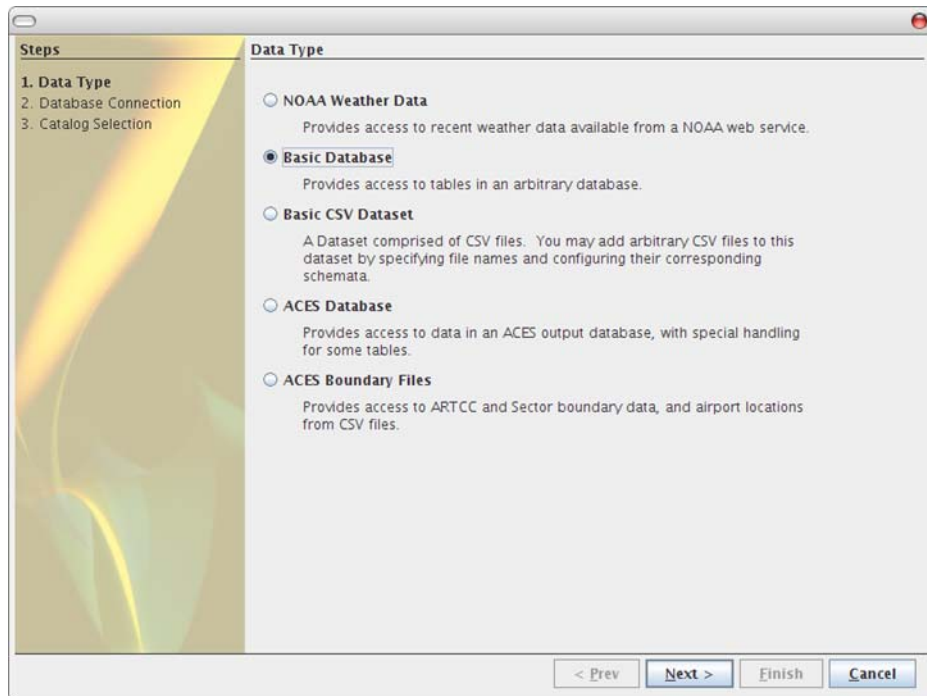


Figure 94 - Dataset Wizard Step 1: select the type of data to load.

Subsequent pages depend on which type of data is being loaded. For example, if the user wants to load database data, they must specify database connection parameters (Figure 95).

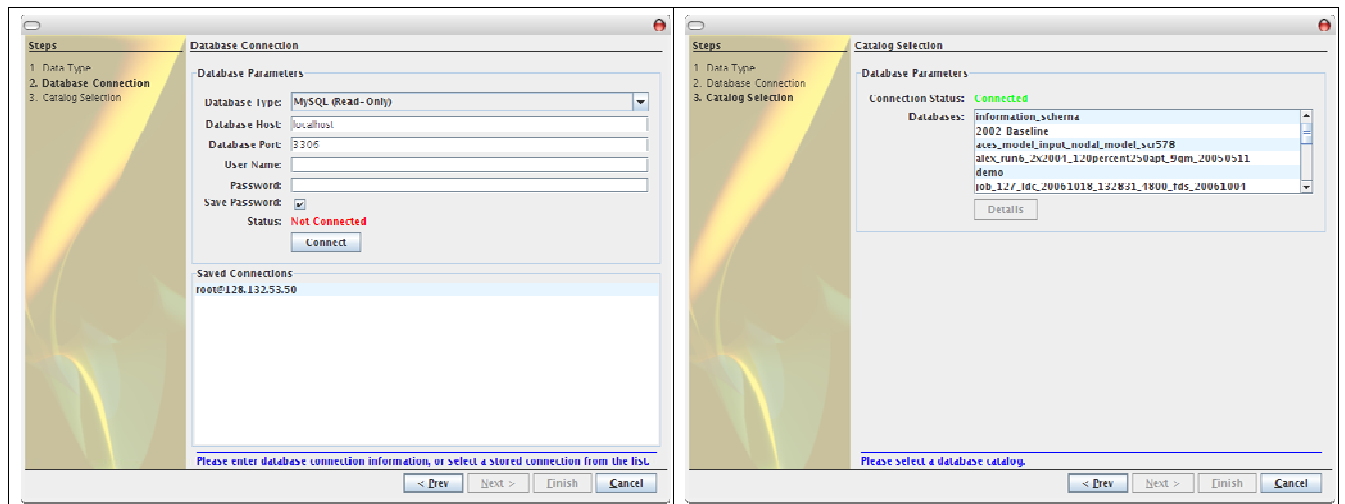


Figure 95 - The Basic Database Dataset Wizard Steps

Once the user has completed the wizard steps, they are given an opportunity to save the dataset definition, so it can be loaded at a later time without going through the wizard again. The Datasets task pane is updated to include the Data Tables that were loaded through the wizard once it is completed.

4.4.5.4 Scene Display

The visualizations created in the Composition Graph are displayed in one or more Visualization Windows. Each Display Container present in the visualization is displayed in a panel in the Visualization Window. Users can add multiple Display Containers to one window (Figure 96), or open each Display Container in a separate window (or any combination of the two). Display Containers can be dragged with the mouse to change their positions within the window or dragged to other windows.

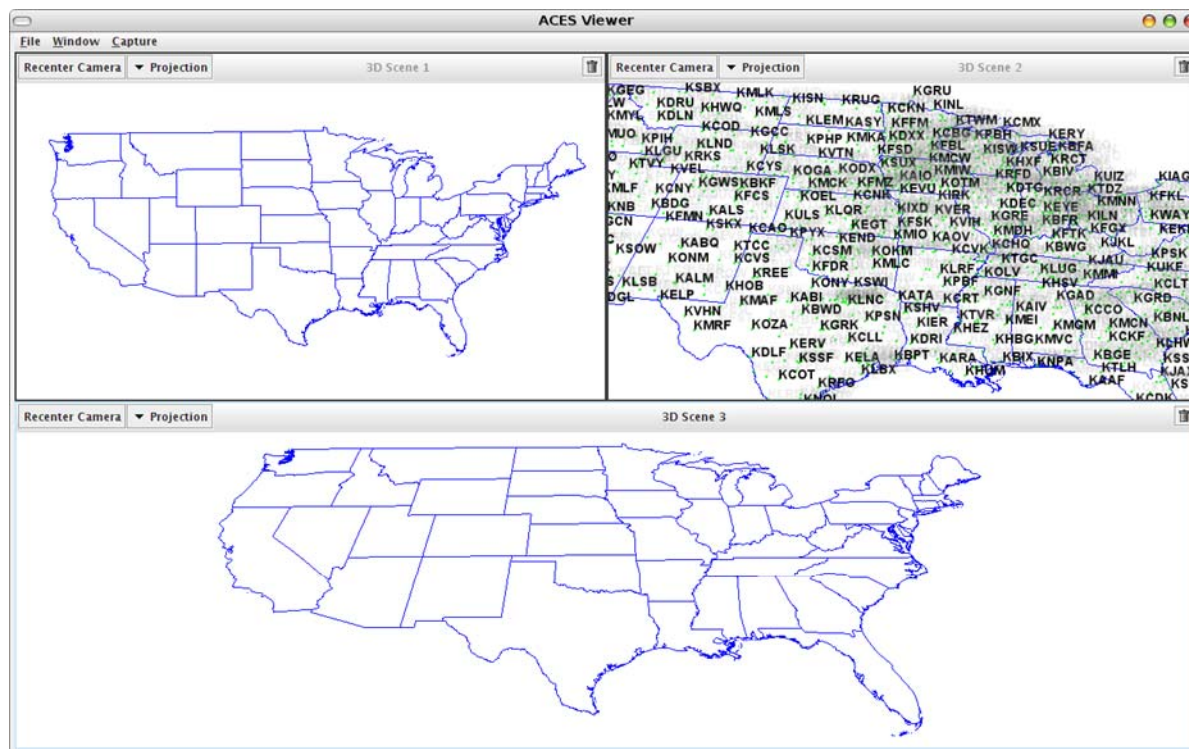


Figure 96 - The Visualization Window

Each Display Container in a Visualization Window has its own toolbar. Each toolbar contains a name for the Display Container (which can be referenced with the Display Container's representation in the Visualization Graph), and a button to close the container. When a Display Container (or its containing Visualization Window) is closed, it is immediately removed from the Visualization Graph, along with any renderers associated with the display. The toolbar also contains user interface controls specific to each Display Container type. For example, the 3D scene provides controls to change the map projection used for the scene, and to re-position the camera so that the entire content of the scene is visible.

4.4.5.5 Movie Capture

Movies can be recorded from the content of the Display Containers in a visualization window. The content of all of the Display Containers in the window will be incorporated into a single movie. Users initiate the movie capture process from the menus at the top of each visualization window. The movie capture controls are added to the bottom of the window (Figure 97).

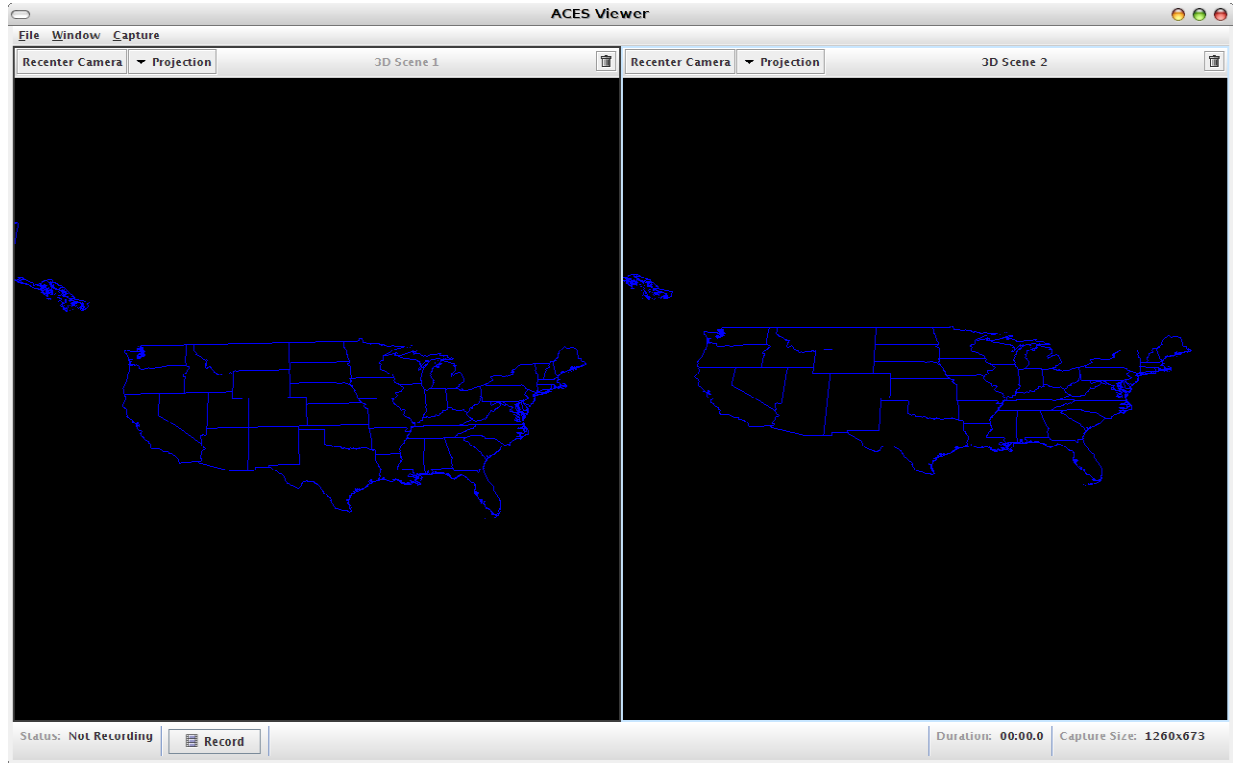


Figure 97 - The Visualization Window after activating Movie Capture

On the right side of the Movie Capture Controls, the size (width, height) and duration of the recorded movie are shown. On the left side there is a field showing recording status, and a *Record* button used to initiate the recording process. Recording does not begin until the user presses this button. Pressing the *Record* button opens a dialog window so that you can specify the parameters of the movie (Figure 98).

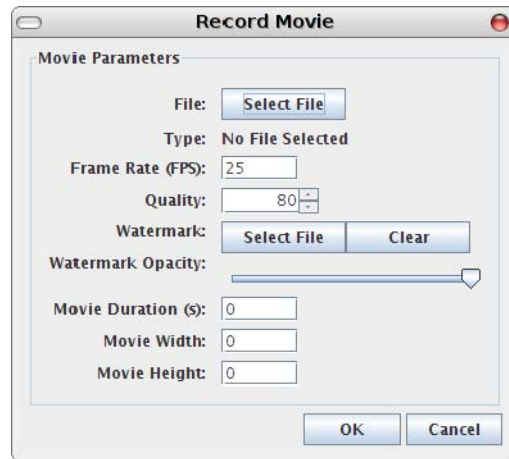


Figure 98 - Movie Parameters Dialog Window

Once the capture process is started, users can interact with the visualization and even make changes to the content while the movie is being recorded.

4.4.5.6 Log Viewer

CACI developed a simple integrated log viewer (Figure 99) to allow users to view logged information for the running application. If a user experiences a problem or error with the application, the log viewer can be opened to view log messages as they are generated by the application. This is also extremely useful during application development. Features of the log viewer include:

- **Generic implementation:** The viewer is a simple Java Swing based component that can be easily added to any application using the Log4J logging system.
- **Filtering by log level:** Log messages can include anything from fatal application errors to low-level application flow trace information. The viewer supports filtering to limit the number of messages users will see.
- **Color Coding:** Two colors are used for each log message, a green or red box indicating whether or not the message indicates an error, and a second color indicating the log level of the message (differentiating, for example, warnings from debugging messages).

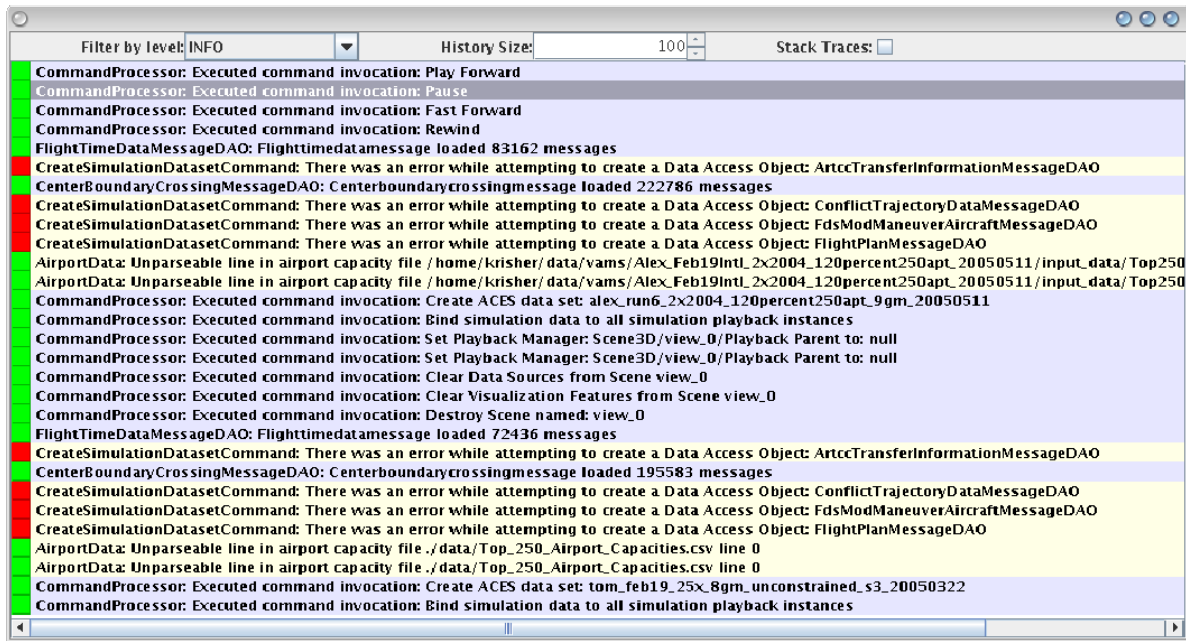


Figure 99 - Integrated Log Viewer Window

4.5 Audit Trail Viewer

The Audit Trail Viewer is a few on few engagement analysis application used for viewing simulation outputs. The Audit Trail Viewer was born as a replacement for IVIEW 2000 and supports visualization replay for many different modeling and simulation data file formats. This part of CACI's JView support contract includes the maintenance and enhancements of the Audit Trail Viewer. The Audit Trail Viewer provides CACI and AFRL engineers the ability to explore supporting larger simulation environments and unique customer visualization requirements.

4.5.1 Graphical User Interface Improvements

4.5.1.1 Digital Terrain Elevation Data (DTED) Panel

The DTED panel was drastically changed to incorporate a new way of loading DTED. In earlier versions of the ATV, in order to load DTED, it was required to have an .atvdtd file along with the simulation file. This .atvdtd file held information like DTED directory, RPF directory, DTED Level, Lower Left Longitude and Latitude. In the new version of the ATV, all of the options for loading DTED are on the panel. The lower left latitude and longitude components were added as well as a DTED directory button. The .atvdtd file is not mandatory to load DTED terrain but is optional for quick loading. The .atvdtd file pre-selects all the options for a simulation. The new change can be seen in Figure 100.

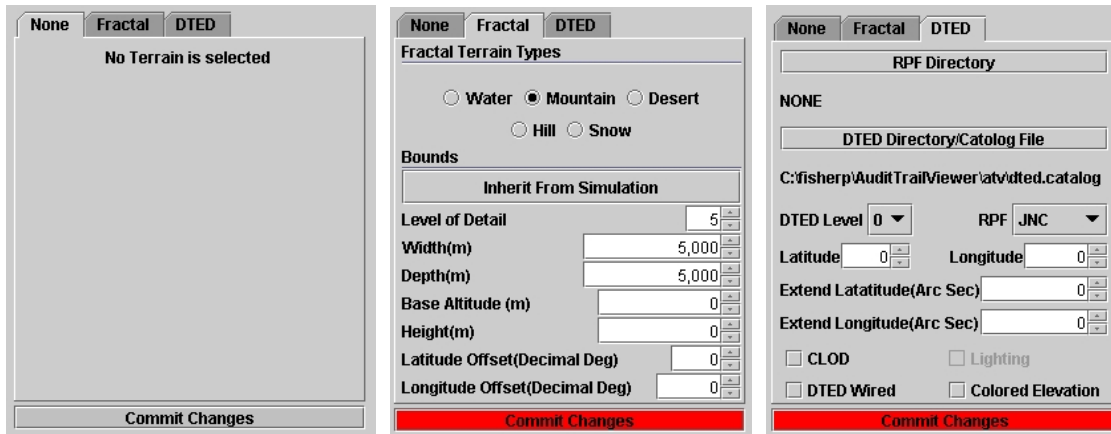


Figure 100 - Terrain Panel

4.5.1.2 Analysis View Tab

CACI has made many changes to the Analysis Viewing Tab under this contract. These changes added new functionality and organized the panel into a cleaner more professional application. Figure 101 shows the analysis view from where we started to where we ended.

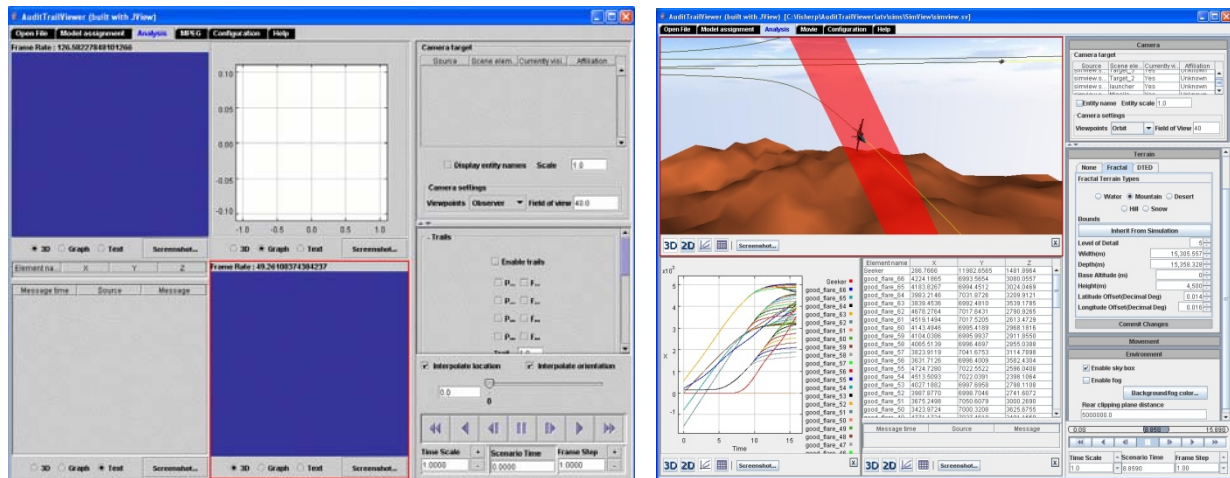


Figure 101 - Analysis View (Old) / (New)

4.5.1.3 Help Panel

The Help Panel was completely changed using Java Help to design a useful and helpful interface with searching capabilities and utilities like printing and backtracking. The old Help Panel was just a collection of html documents that users could read though and there was no searching or structuring. Figure 102 shows the interface changes from the old to the new Help Panel.

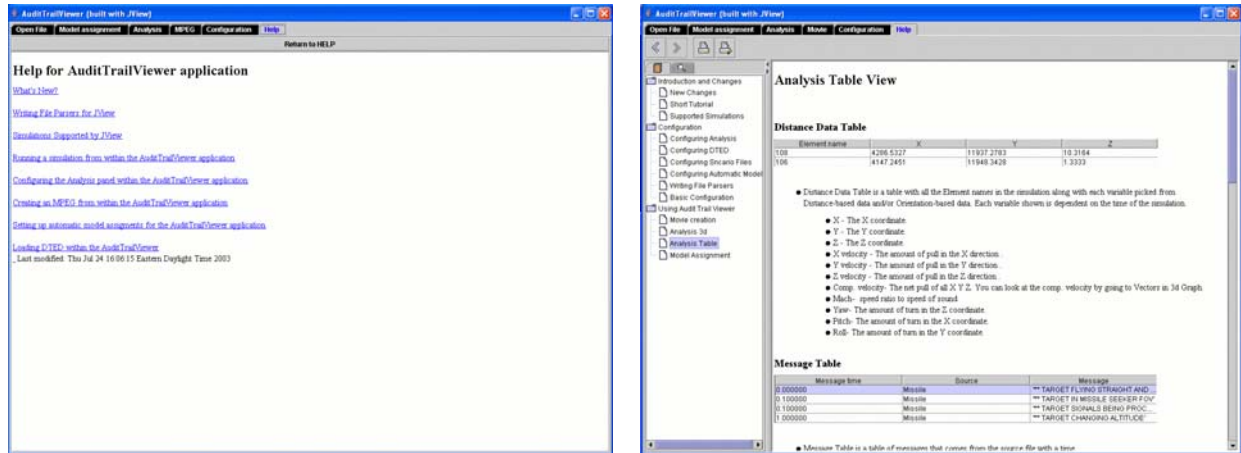


Figure 102 - Help Panel (Old) / Help Panel (New)

4.5.1.4 Geometry Panel

The Geometry Panel is the second panel that the user sees in the ATV. CACI enhanced the Geometry Panel by incorporating the element viewer. The element viewer tool can be used to view and select elements such as aircraft in three different ways (Large, Small or Iconic). A popup mouse over view was added to the element viewer so the user can get more detail on the model. The popup now includes a 2D version of the image on the lower right corner of the popup. The new element viewer tries to fit as many images as it can with a minimum size preference. The element viewer, seen in Figure 103, gives the ATV a professional look as well as an easy way to identify which model the user wants.

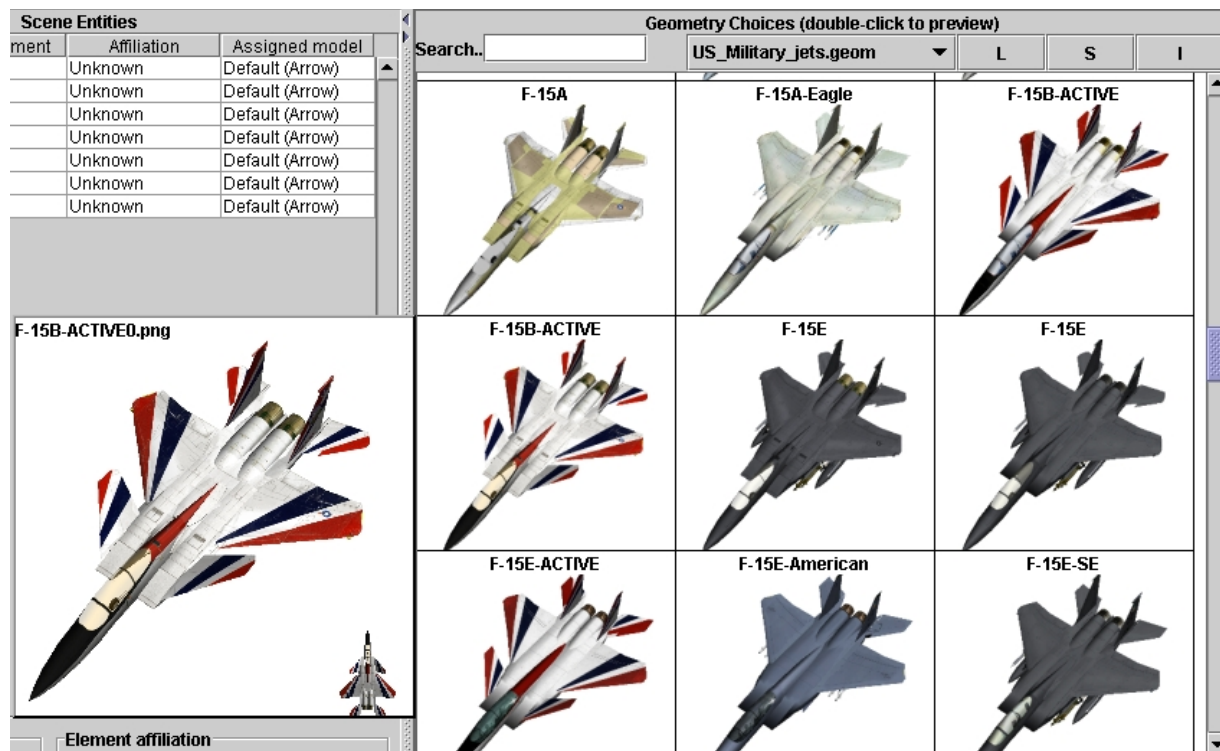


Figure 103 - ATV Element Viewer

4.5.2 2D Graphical Engine – 2D View

The 2D graphical Engine is the largest feature that CACI provided in the ATV. There are many users of the ATV that only need a 2D God's eye view of the simulation run.

4.5.2.1 Arrow2D

CACI developed the Arrow2D to be used as the default 2D element in the ATV as well as a general 2D element in JView. The arrow class is configurable to make almost any type of arrow imaginable. Length and width can be modified to set the boundary of the Arrow2D. Colors can be picked to change the color of the boarder or the Arrow2D. Also, the trail width and length can be adjusted by changing the ratio of each. By adjusting the tail length you can inversely change the Arrow2D head length and angle. Figure 104 shows a diagram of the Arrow2D object.

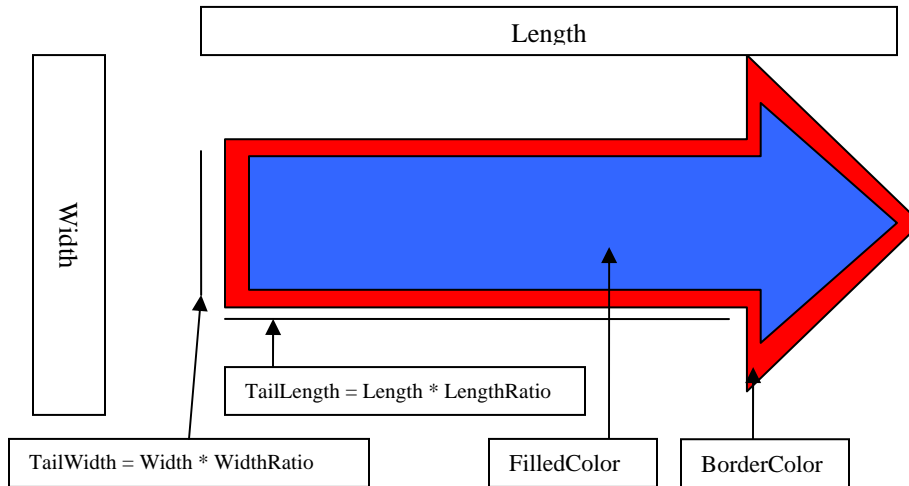


Figure 104 - Arrow 2D

4.5.2.2 Trail 2D

A 2D element to display movement paths was created mimicking the 3D version. The trail future and past is created in four parts. Part one is the past trail from beginning to current element point. The second part of the trail is created from the current element point to the interpolation point of the element. An interpolation point is a point where the time of the element is in-between two known times. Part three is the line segment from interpolation point of the element to the next element point. The forth and last line segment is from the next element point to the ending point. Figure 105 displays a breakdown of the 4 parts of a 2D Trail.

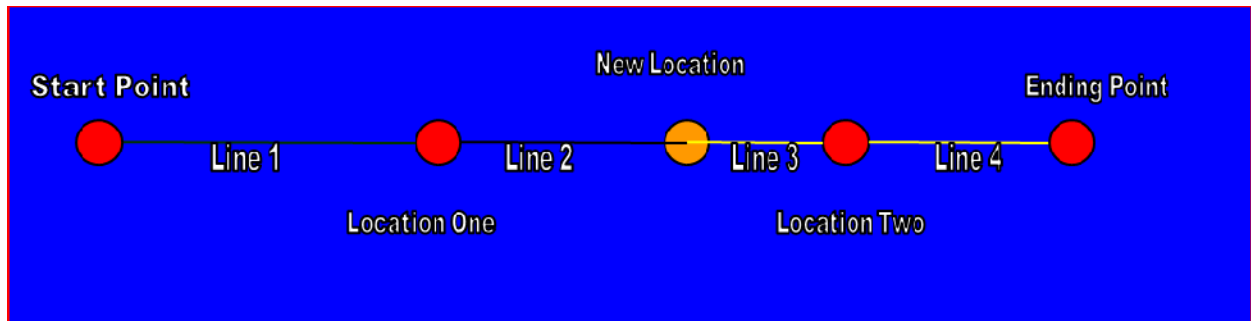


Figure 105 - Trails 2D

4.5.2.3 Cone2D

Cone2D was created to represent radar beams from elements such as seekers. Cone2D is defined by values of yaw and pitch. Pitch determines the length as well as the curvature of the cone. A pitch of 0 would look like a triangle and a pitch of 180 would look like a circle. The yaw value determines where the cone should be pointing. Figure 106 displays an example of two Cone2D's on a single aircraft.

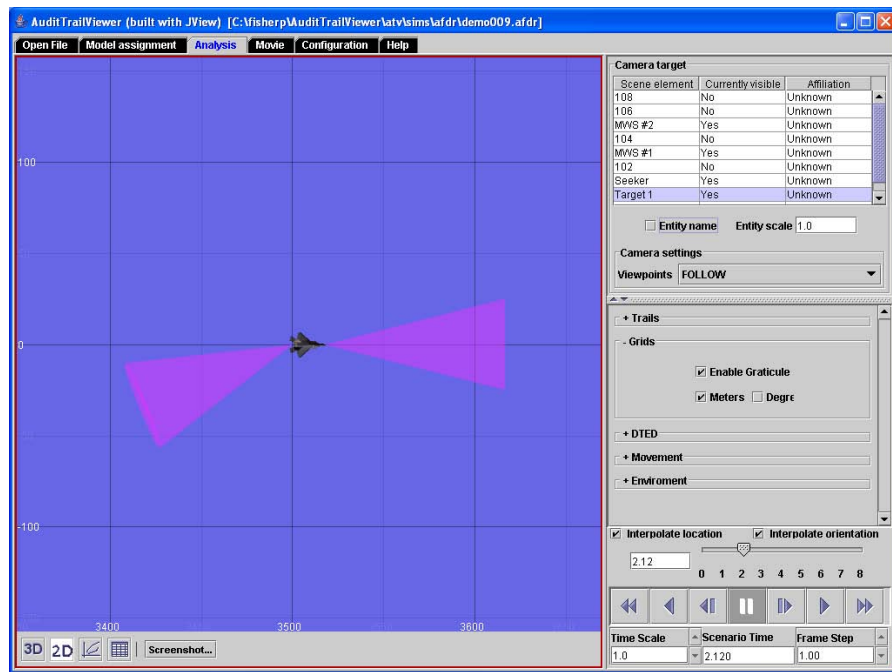


Figure 106 - ATV Cone 2D

4.5.2.4 Models 2D

2D Models were added to help users see information faster and clearer. All models beside the cone and sphere are raster image models. Raster image models are images that don't get scaled from zooming in and out. If no image has been selected, a default arrow is drawn. Models can be scaled by entering a value in the entity scaled box which will be multiplied by the default scaling factor. By default, this is set to 1.0 with the models being a 50x50 pixel image resulting in all models displayed in the same size. Figure 107 displays an example with several 2D models.

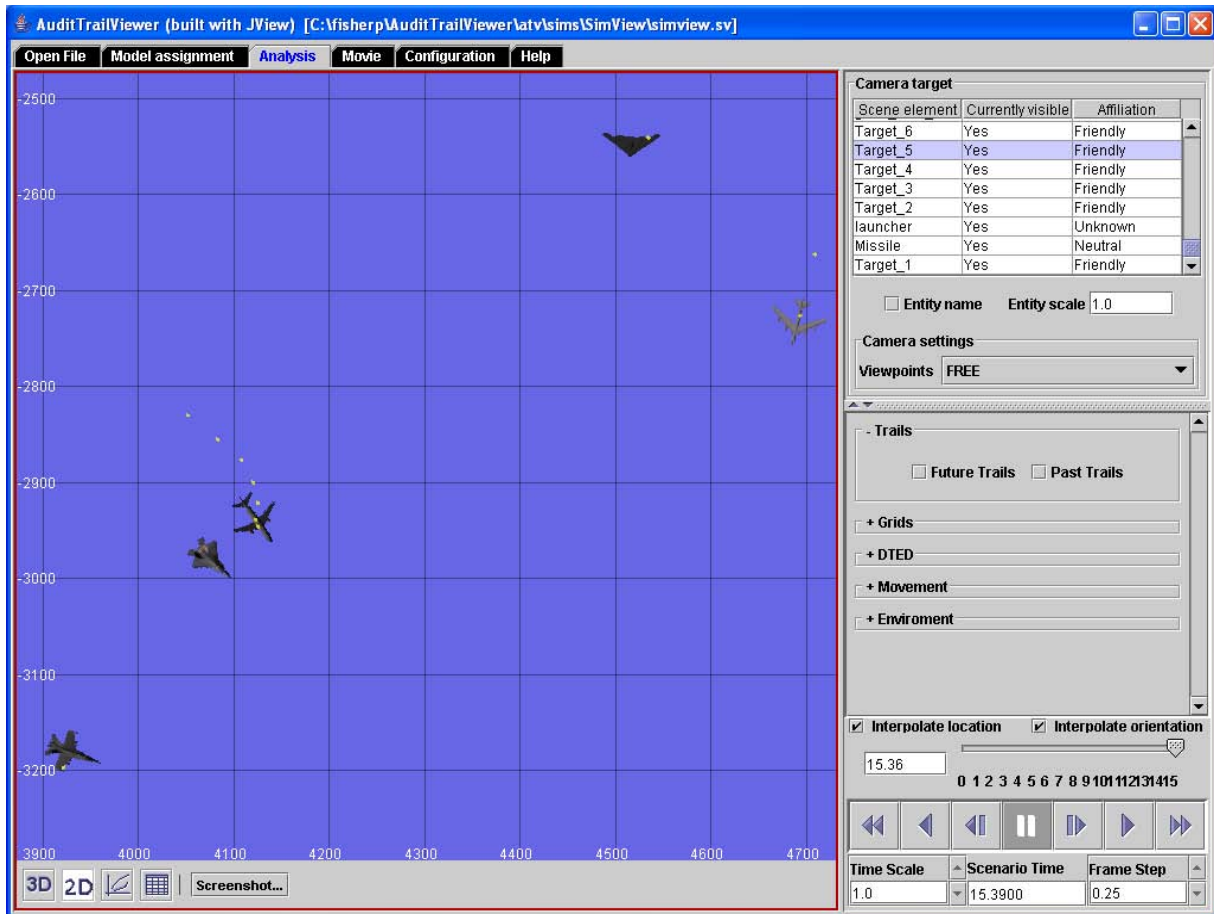


Figure 107 - ATV 2D Models

4.5.2.5 Labels 2D

CACI created a new 2D Labels element and a different camera setting had to be created. For the 2D camera modes, it was necessary to create a `MovementManager2D` to help with assignment of camera modes. There are two camera settings for the 2D View which are *Follow* and *Free*. The *Follow* mode follows the element that is currently selected and the user cannot move away from the object. For the *Follow* mode, the `BaseSceneElement2D` was modified to implement a listener that would respond to 2D element movements. This was the only way to check if the element would move so that camera could be updated on follow mode. The *Free* mode is a camera mode that can both move and zoom. There are no attachments with *Free* mode. Figure 108 shows an example with 2D Labels.

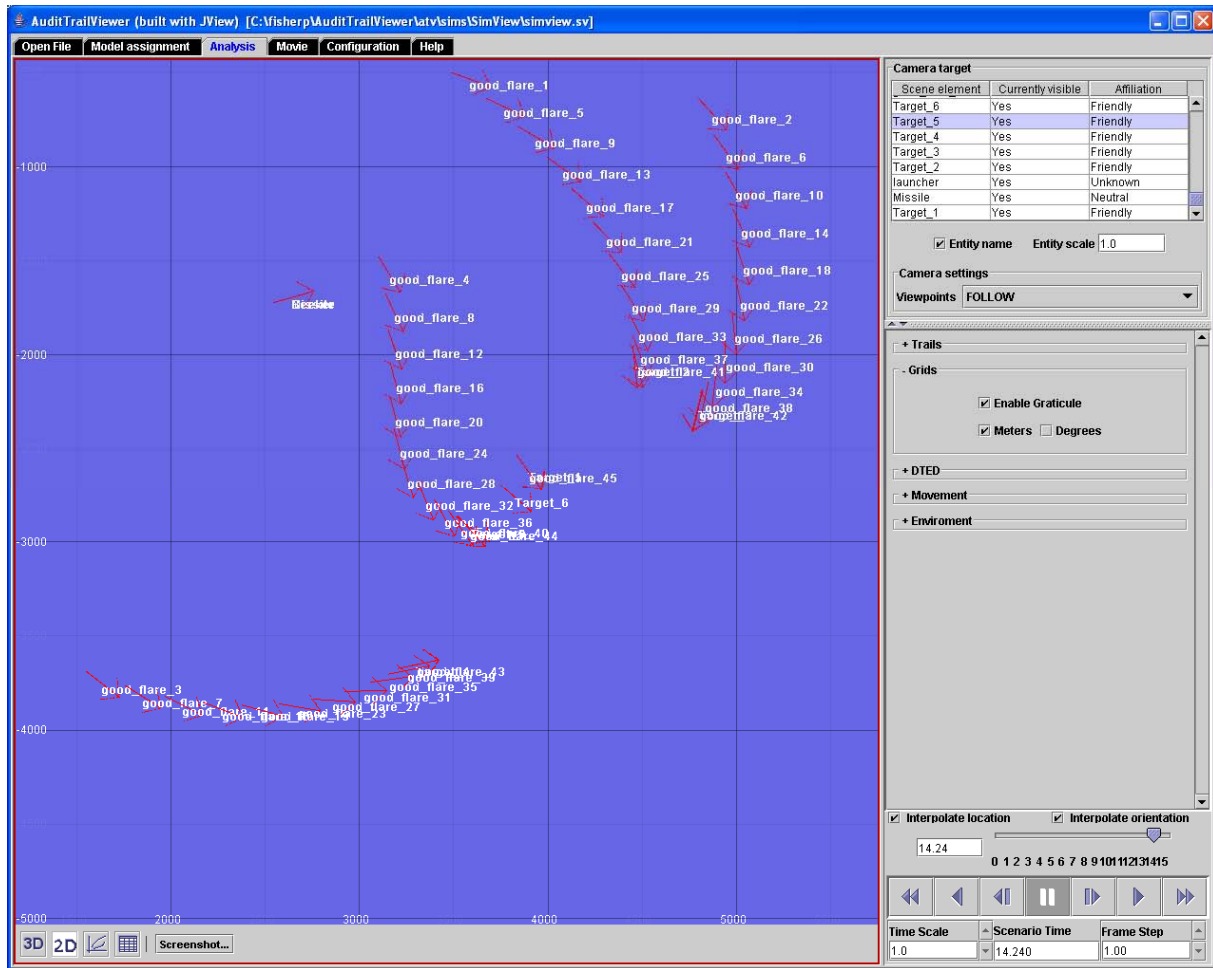


Figure 108 - ATV 2D Labels

4.5.2.6 Imagery 2D

As with the 3D display, the 2D display includes the ability to add imagery to the background. JView already contained support for image generation for several data formats (RPF and DTED are examples). Unlike the 3D display, no underlying mesh needs to be created and imagery may be added without the presence of DTED data. Figure 109 shows an example of JNC RPF data blended with a DTED image.

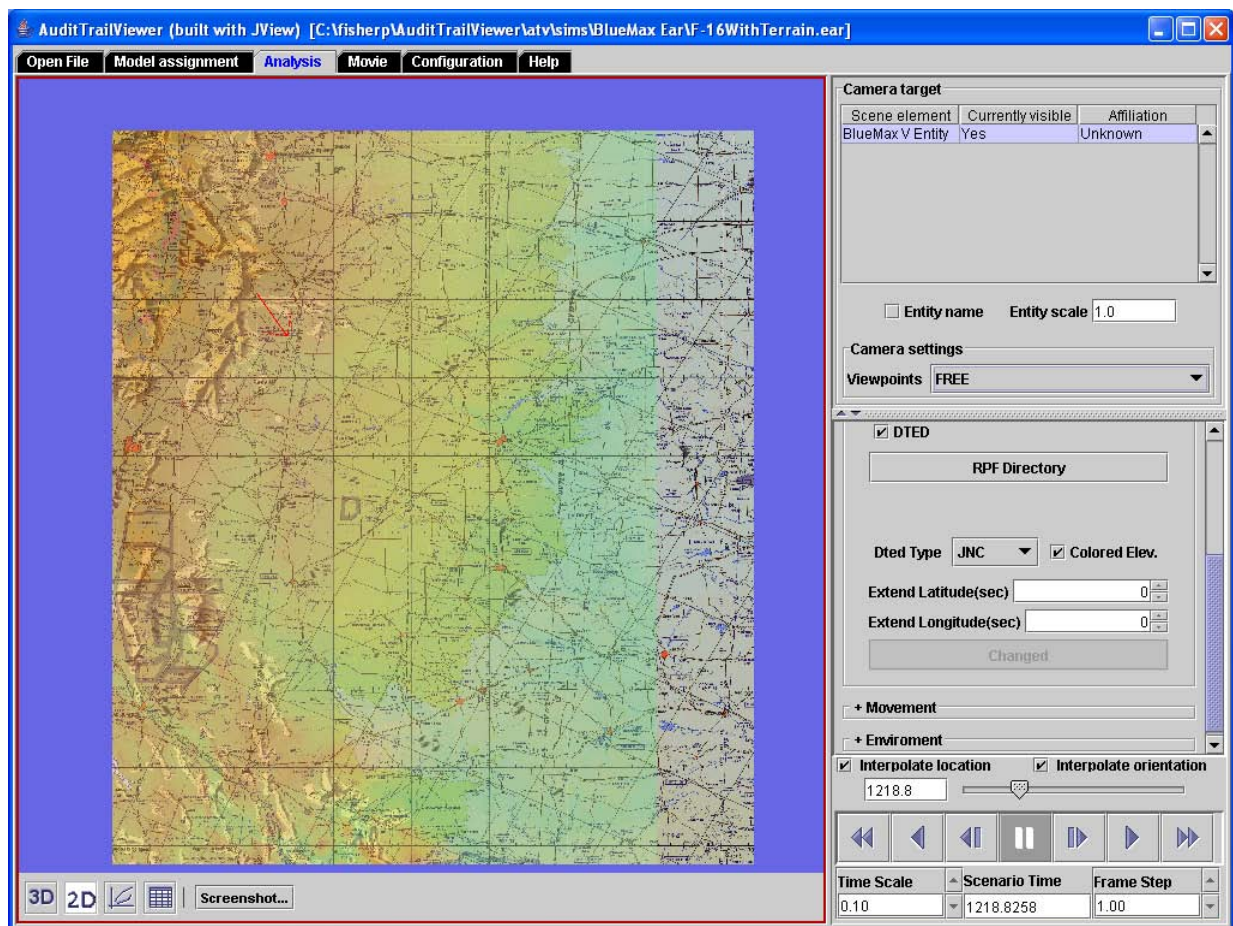


Figure 109 - ATV with 2D Maps

4.5.3 Splash screen

Using GIMP, the CACI team and the JView government staff developed the new splash screen as seen in Figure 110. This picture shows one of the new models that can be used. It also displays the 2D view and table view, and is an improvement over the previous splash screen.



Figure 110 - ATV Splash Screen

4.5.4 Representation of Ground Entities

Ground entities are difficult to represent in the Audit Trail Viewer. The Audit Trail Viewer loads a simulation output file that has multiple entities and contains location data at an update interval. The ATV entities are placed according to the time of the simulation. If the time of the simulation is equal to a time in the simulation file then the location associated with that time is used, otherwise an interpolated time is calculated. This works well with airborne models but gives an unrealistic view for ground elements. Figure 111 displays an example of what an entity might do on the ground. The black line represents the ground and the red line represents an entity trajectory. Interpolation of the ground unit position between reported locations results in a path that passes through the terrain in several locations.

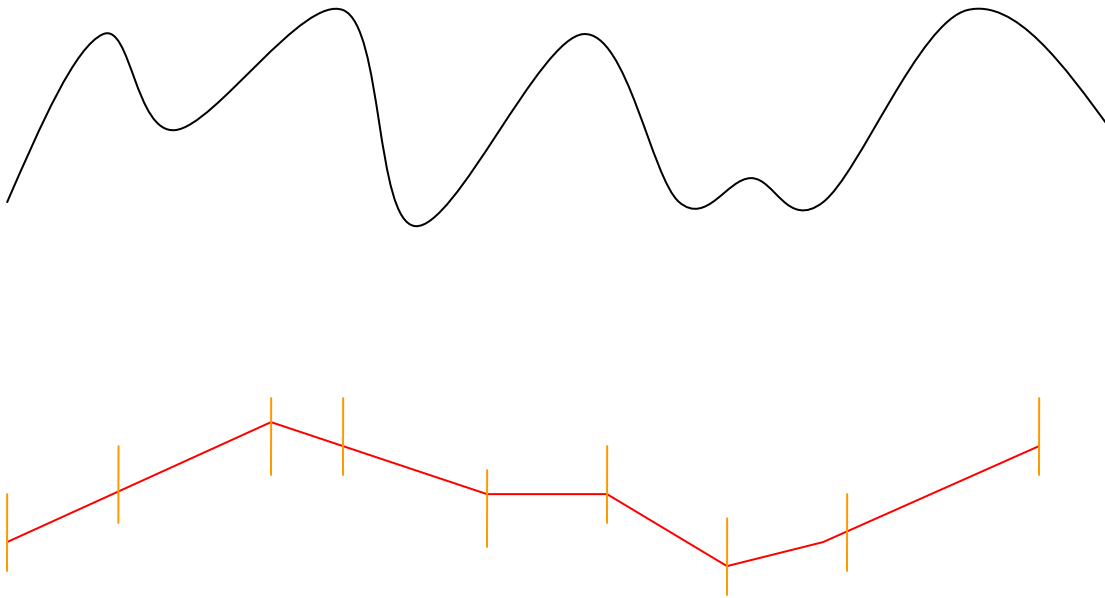


Figure 111- Ground Trail

To accurately represent an entity at ground elevation, we first need to know the elevation of the ground. In JView, Digital Terrain Elevation Data (DTED) is drawn as triangles that use DTED elevation posts as the triangle vertices. If the ground point is very near to a post, then the elevation of the post is used as the actual elevation. Most of the time elevation is calculated by finding what triangle the point lies on and then interpolating the elevation of the point using the three elevation posts of the triangle. Figure 112 shows an example of a point on a triangle strip. Selection of which triangle a point lies on is determined by calculating the horizontal percentage offset and vertical percentage offset from the lower right post. If the sum of the horizontal and vertical offsets is greater than one, then the upper left triangle is used for interpolation, otherwise the lower right triangle is used. Once the triangle is selected, we interpolate the elevation of the point by using the x offset and y offset for that point.

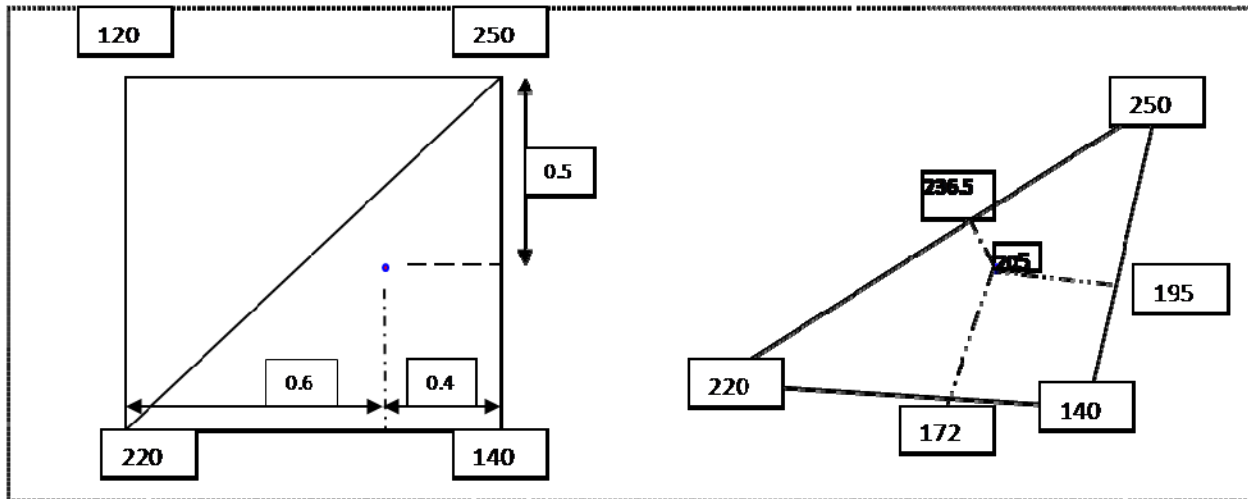


Figure 112 - Triangle Selection Example

4.5.4.1 Highlight 2D Elements

CACI added the ability to highlight 2D elements to solve the problem that terrain can look very similar to the 2D elements and make it hard to identify an object. Highlighting can also be used to identify the current 2D element selected after a mouse click or other operation. As seen in Figure 113, it is easier to see elements that are highlighted. A custom option panel was added in the ATV to the environment portion of the analysis section to support highlighting of element. These options can choose from a range of colors to highlight the elements or choose the colors by affiliation.

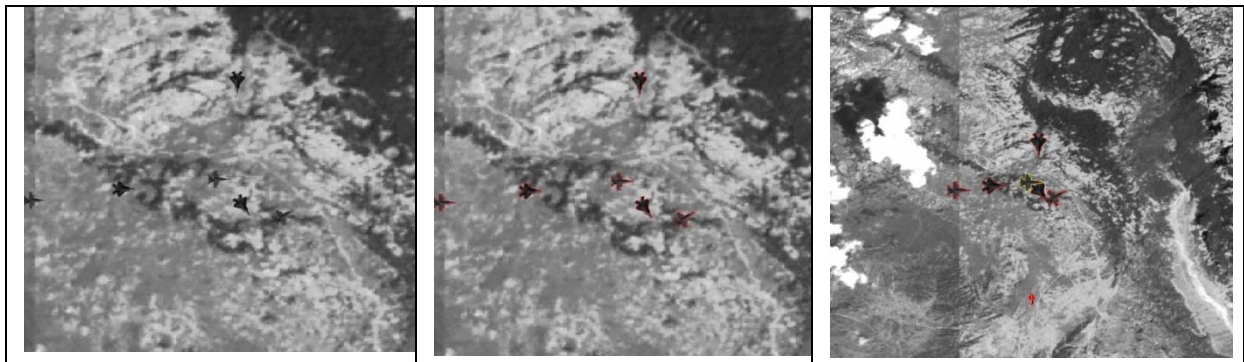


Figure 113 - Highlighted Elements

4.5.4.2 3D Views

To show information in a different perspective, we added two new viewports into the ATV. The first viewport, the *Cockpit View* is simply, a view from the cockpit of the aircraft (not all aircraft have imagery for the cockpit and on these instances this view cannot be selected). The cockpit

view is shown on the left and middle image of Figure 114. The second new viewport is called *Heads Up Display (HUD) View*. CACI developed a basic HUD display that shows heading information as well as elevation angles. The HUD view is shown in the right most image of Figure 114. This HUD view has a great deal of potential to display more information in the future.

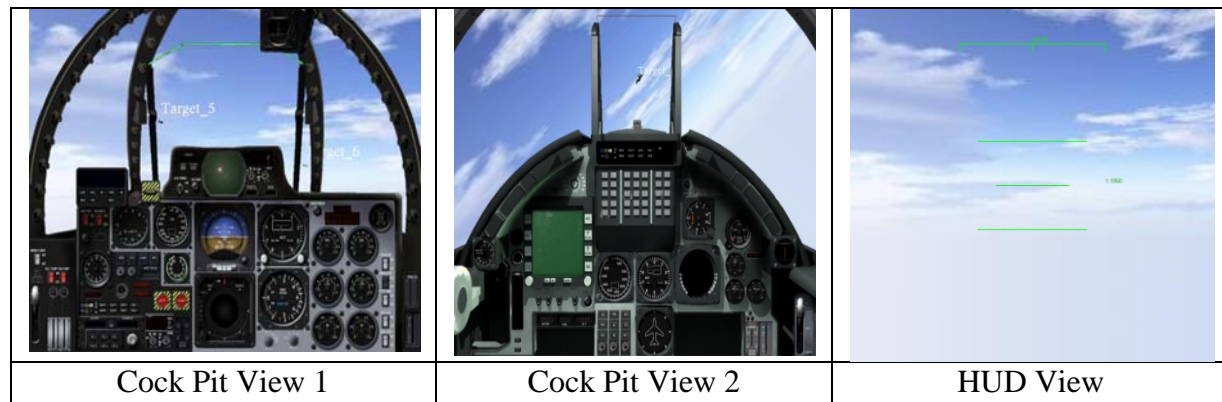


Figure 114 - HUD and Cockpit Views

4.5.4.3 GATER II Simulation

CACI added support for the GATER II format which is comma delimited and has options to show the affiliations, radar information, sound and above ground level (AGL) elevation. CACI also developed a loader to view GATER II simulation files in the ATV. Keyholes, which are zones within a radar system's coverage that are being jammed by another entity, needed to be created to support the GATER II output files. GATER II allows for any one radar site to have multiple keyholes. The shape of a keyhole is an oblong conic shape and Figure 115 shows an example of 2 keyholes displayed in the ATV.

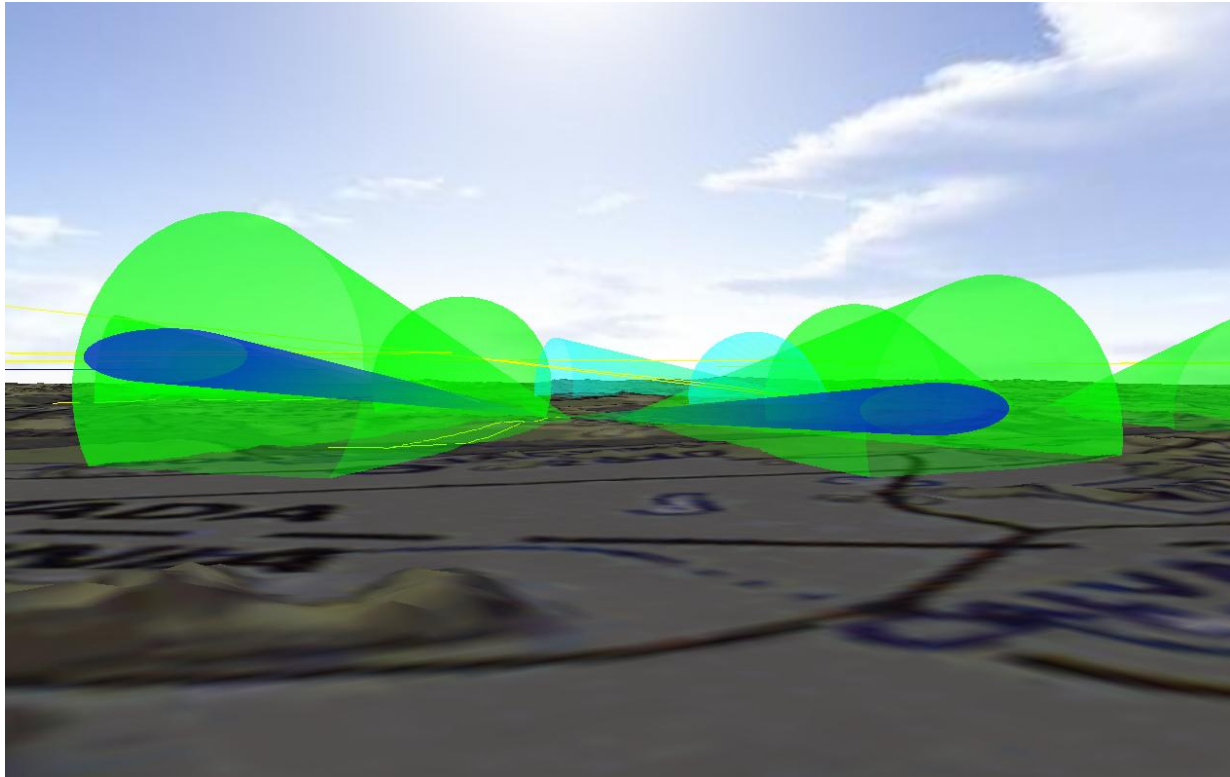


Figure 115 - ATV Keyholes

To identify targets that have been detected or acquired, CACI colored the existing models to represent the current state of an element. Elements that are red are detected, elements that are yellow are acquired and elements that are orange represent both. Figure 116 shows an example of 3D model highlighting.

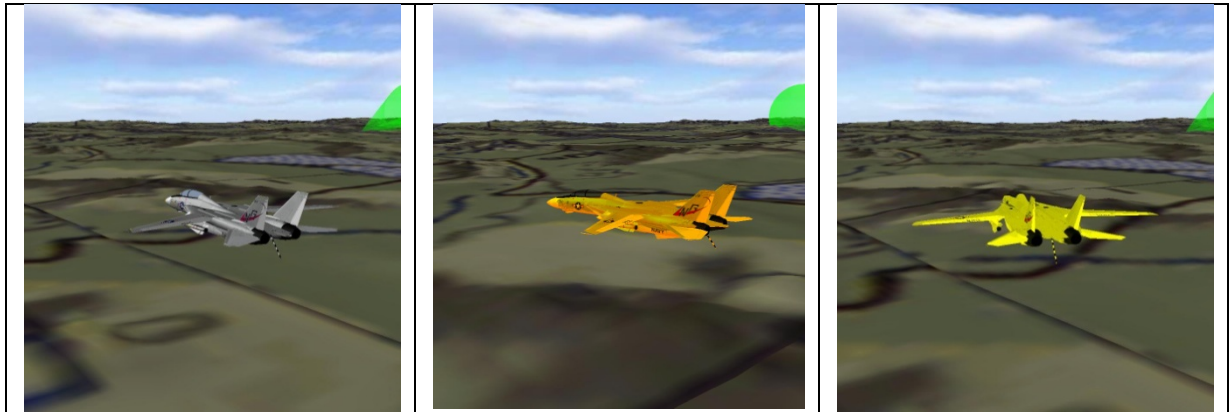


Figure 116 - Identifying ATV Elements

4.5.4.4 ATV and the World

CACI integrated the JView *World* into the Audit Trail Viewer. A new mouse/keyboard interaction model was created called the *SurfaceNavigator* and a small but useful Graphical User Interface (GUI) to interact with the *World* was added to the ATV. Figure 117 shows the ATV with the *World* and also the GUI that was implemented to control *World* settings.

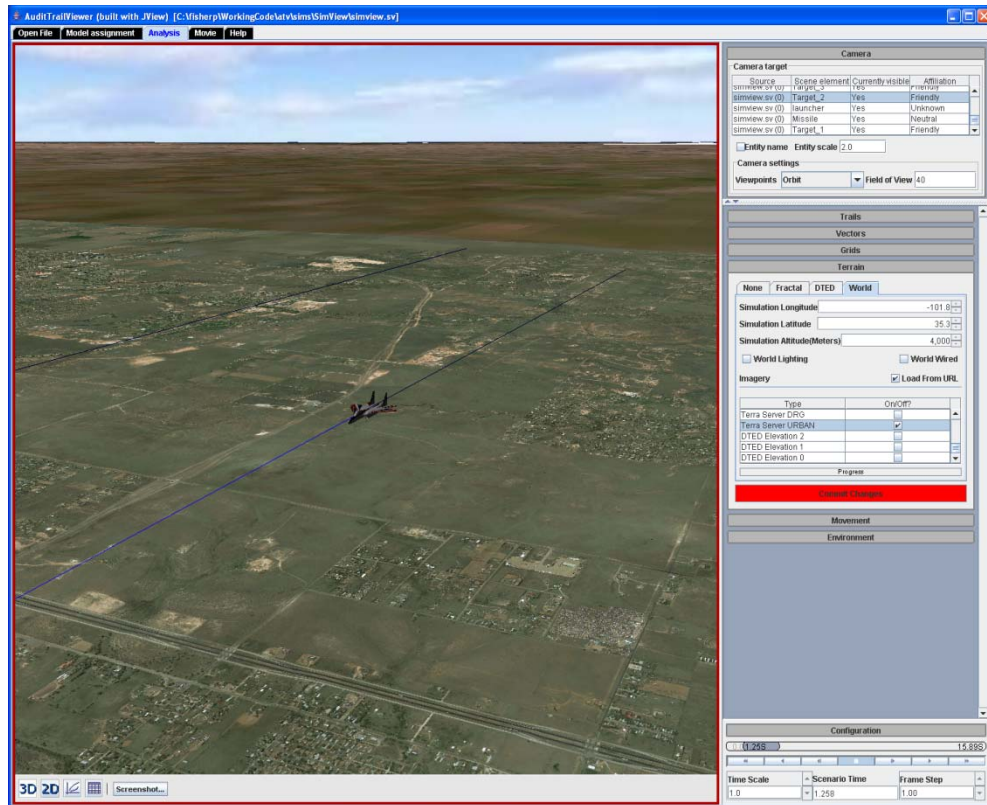


Figure 117 - ATV *World*

The *SurfaceOrbitNavigator* was added to the ATV not just to provide the correct control of the *World* but also to provide a better mouse/keyboard movement tool than the previous one. This new navigation system allows smoother movement while orbiting or panning. The *SurfaceOrbitNavigator* was added to the Orbit, Observational, and the Movie Viewports. The keyboard/mouse controls were changed to better interact with the navigator style.

4.6 3D Model Development

Development of a library of 3D models in the Open Flight format is to be owned by the Air Force and distributed under a GOTS license to JView users.

4.6.1 COTS Software Used

4.6.1.1 3D Studio Max (3DS)

The primary software used to develop the model library was 3D Studio Max (3DS). CACI developed the meshes and overlaid the textures using this program. Each model was delivered first in .max format. The models were then exported using additional COTS software.

4.6.1.2 PloyTrans

CACI was provided with a copy of PolyTrans to export the 3DS model files to the Open Flight (flt) format and obj formats. PolyTrans is a scene translation tool and a 3d geometry toolkit. This program was found to have several problems exporting files containing articulation. It was determined from these communications that PolyTrans would not be an acceptable solution for creating flt models. CACI researched other possible tools which led to the purchase of Multigen Creator.

4.6.1.3 Multigen Creator

CACI was provided with a copy of Multigen Creator in order to resolve the issues PolyTrans had exporting articulated models from 3DS to open flight. Creator only became available near the end of this contract and as a result, only a few models were actually completed using this software. Section 4.6.3.1 discusses these models.

4.6.2 JView Model Software Developed

4.6.2.1 ElementViewer

CACI enhanced JView's Drag and Drop Element Viewer to display axis, unit's values and file names. This allows users to see the models as JView will display them. Figure 118 shows an example of a model displayed in the Element Viewer.

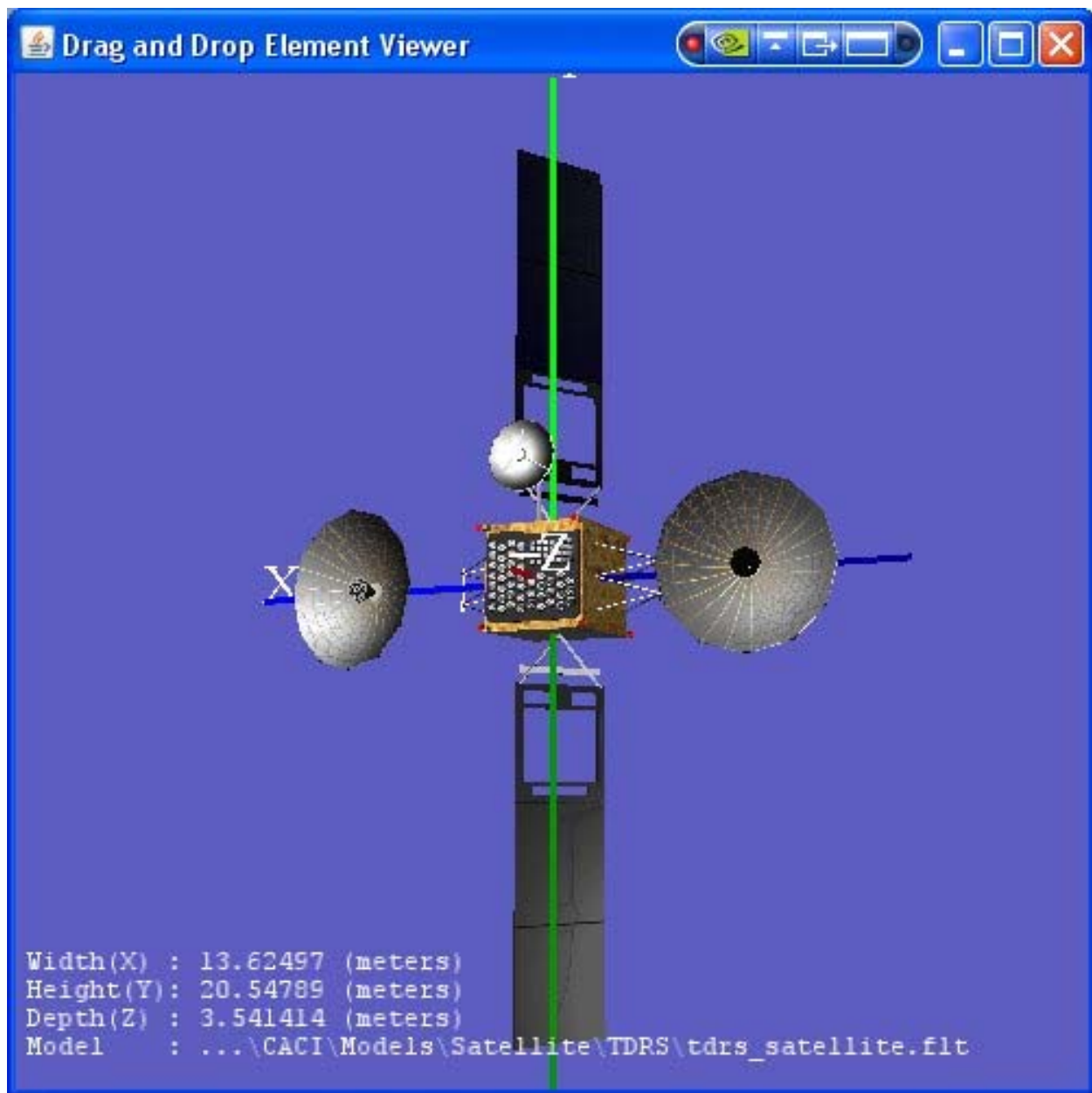


Figure 118 - Element Viewer

4.6.3 Model Library

4.6.3.1 Models Developed

Thumbnails of the models developed by CACI under this contract are shown in Tables 2 through 13. Each model directory contains the model in 3 different formats, .max, .flt, and .obj.

4.6.3.2 Articulated Models Developed

CACI only received MultiGen Creator near the end of this contract and as a result, only a few of the models have had articulation added. In the next section, the AWACS thumbnail (Table 2, left), predator thumbnail (Table 5, upper right) and Cell Phone model thumbnail (Table 9, lower left) contain articulated components.

4.6.3.3 Model Thumbnails

CACI developed a total of 33 models on this effort and their thumbnails are shown in Appendix A.

5.0 Conclusion

This final technical report illustrates CACI's progress in developing, implementing and integrating visualization technologies that support NASA and the Federal Aviation Administration (FAA). Specifically, CACI provided direct support to JView users, developed the JView World object (resulting in a joint-AFRL/CACI patent), broke new ground in terms of de-cluttering concepts, developed ATV version 1.2, developed a library of 3D models, and developed a new application using JView technology called the Airspace Concept Evaluation System (ACES) Viewer that provides visual representations of the output of ACES. We count it as a great success that JView represents a standard visualization solution and provides an environment that adapts to user needs vice forcing them to adapt to the tool. It has been a privilege to partner with AFRL and to get to build on the impressive foundation created by AFRL/RISF.

REFERENCES:

[1] Moore, J., and McVay, A., "Method for Loading and Displaying Massive Gridded Datasets Patent #60/879,211", 2007.

[2] National Aeronautics and Space Administration, "World Wind", <http://worldwind.arc.nasa.gov>, 2006.

APPENDIX

Table 2 - Aircraft / Surveillance



 <p>A 3D rendering of an AWACS (Air Warning and Control System) aircraft, specifically an E-3 Sentry. It is a four-engine jet with a large, black, dome-shaped radar sensor mounted on its upper fuselage. The aircraft is shown from a three-quarter front view, with "US AIR FORCE" visible on the side.</p> <p>AWACS</p>	 <p>A 3D rendering of an RC135 (Rivet Joint) surveillance aircraft. It is a four-engine jet with a long, slender fuselage and a large, black, rectangular sensor pod mounted on its upper fuselage. The aircraft is shown from a three-quarter front view, with "US AIR FORCE" visible on the side.</p> <p>RC135 (Rivet Joint)</p>
---	---

Table 3 - Aircraft / Bombers

 <p>A 3D rendering of a B2 Bomber, a stealth bomber. It is a dark, angular aircraft with a highly stealthy design, featuring a large, flat, delta-shaped wing and a long, narrow fuselage. The aircraft is shown from a three-quarter front view.</p> <p>B2 Bomber</p>

Table 4 - Aircraft / Fighters





 <p>F16</p>	 <p>F22 (Raptor)</p>
 <p>F35A (JSF)</p>	 <p>F117 (Stealth Fighter)</p>

Table 5 - Aircraft / Unmanned Aerial Vehicle (UAV)




 <p>A 3D rendering of a Global Hawk UAV, showing its high-wing configuration, long wingspan, and V-shaped tail. The aircraft is dark grey with a blue star insignia on the wing.</p>	 <p>A 3D rendering of a Predator UAV, showing its high-wing configuration, long wingspan, and V-shaped tail. The aircraft is light grey with a blue star insignia on the wing.</p>
<p>Global Hawk</p>	<p>Predator</p>
 <p>A 3D rendering of a Rascal UAV, showing its high-wing configuration, long wingspan, and V-shaped tail. The aircraft is white with red accents on the wings and tail.</p>	
<p>Rascal</p>	

Table 6 - Computer Hardware / IO Devices


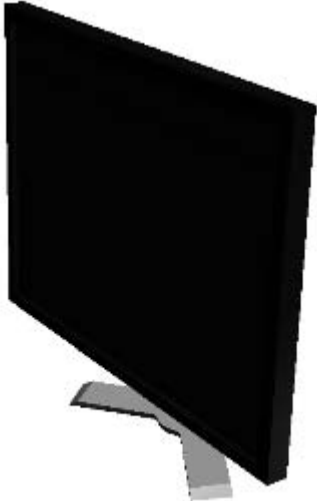

 <p>Keyboard (Dell)</p>	 <p>Monitor (Dell)</p>
 <p>Mouse (Dell)</p>	

Table 7 - Computer Hardware / CPUs



	
Computer Case (BOXX)	Laptop (HP)

Table 8 - Network Hardware



	
Router Wired (Linksys)	Router Wireless (Linksys)

Table 9 - Communications Equipment

	
Cell Tower	Communication Tower
	
Cell Phone	

Table 10 - Ground Vehicles




 <p>2006 Chevrolet Silverado HD Crew (Blue)</p>	 <p>2006 Chevrolet Silverado HD Crew (White)</p>
 <p>Desert Patrol Vehicle (DPV)</p>	

Table 11 - Satellites


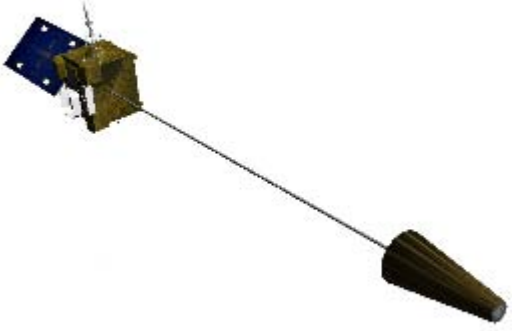
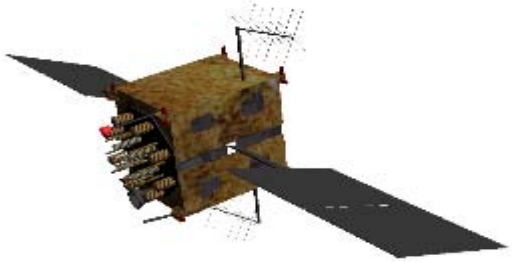
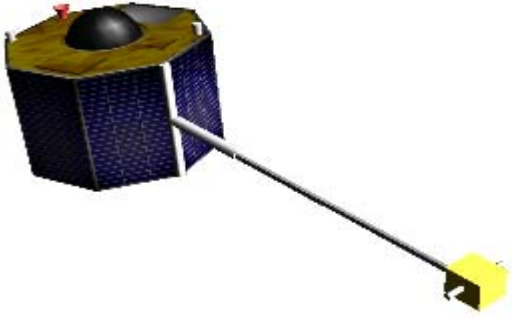
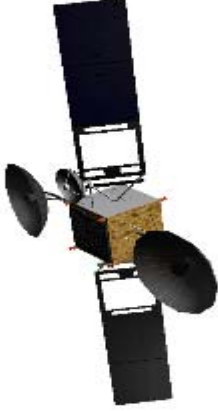
 <p>DSP</p>	 <p>GOES</p>
 <p>GPS IIR</p>	 <p>ST5</p>
 <p>TDRS</p>	

Table 12 - Space Telescopes

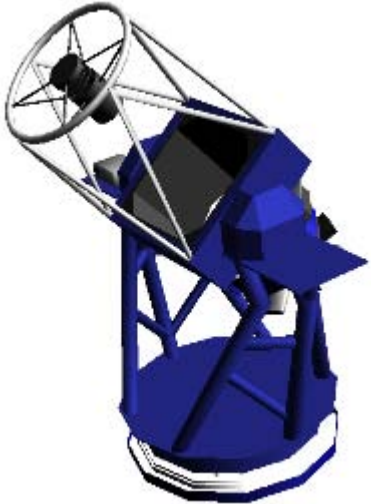


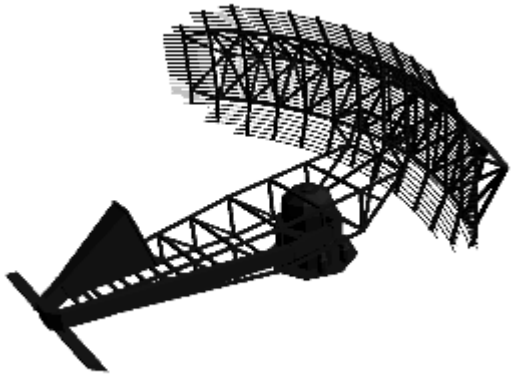
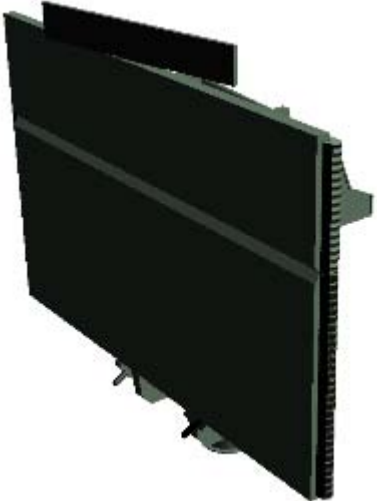
 <p>Faulkes</p>	 <p>Maui</p>
 <p>Sloan</p>	

Table 13 - Radars

 <p>SPS 49</p>	 <p>TPS 75</p>
--	---

ACRONYMS:

ACES – Airspace Concept Evaluation System
AGL – Above Ground Level
API – Application Programming Interface
AWT – Abstract Windowing Toolkit
ATV – Audit Trail Viewer
CADRG – Compressed ARC Digitized Raster Graphics
CIB – Controlled Image Base
CLOD – Continuous Level-Of-Detail
CSV – Comma Separated Values
CUNE – Characterization of the UAV Network Environment
DOQ – Digital Ortho Quadrangle
DRG – Digital Raster Graphic
DTED – Digital Terrain Elevation Data
EDT – Event Dispatch Thread
FAA – Federal Aviation Administration
GRIBB – Gridded Binary
GUI – Graphical User Interface
HUD – Heads Up Display
ILOD – Imagery Level-of-Detail
JDBC – Java Database Connectivity
JNC – Jet Navigation Charts
JOGL – Java OpenGL Library
KML – Keyhole Markup Language
PTR – Screen Pixel to Texture Texel Ratio
NAD83 – North American Datum of 1983
NAS – National Air System
NDFD – National Digital Forecast Data
NOAA – National Oceanic and Atmospheric Administration
OOD – Object Oriented Design
RPF – Raster Product Format
UAV – Unmanned Aerial Vehicle
USGS – United States Geological Survey
UTM – Universe Transverse Mercator
VAMS – Virtual Airspace Modeling and Simulation
WMS – Web Map Service
WGS84 – 1984 World Geodetic System
XML – Extensible Markup Language
3DS – 3D Studio Max